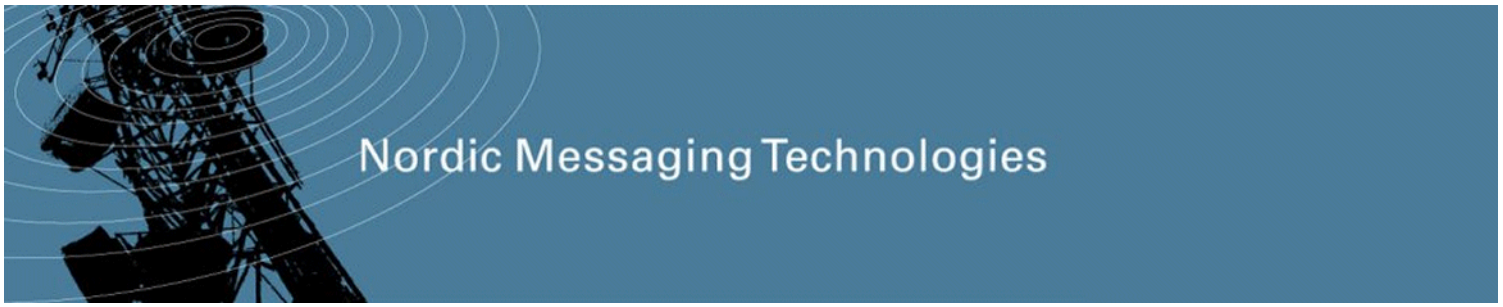

Enterprise Messaging Gateway

User’s Guide - 3.0.16

Nordic Messaging Technologies, Sweden, http://www.nordicmessaging.se, info@nordicmessaging.se

Contents

1. Acknowledgements 12
    1.1 OpenSSL 12
    1.2 LibXML 12
    1.3 PCRE 12
    1.4 PCRS 12
2. Overview 13
    2.1 Licensing 13
    2.2 Messages 13
    2.3 Binary messages and User-Data Header (UDH) 14
    2.4 Long messages 14
    2.5 MMS 14
    2.6 Connectors 14
    2.7 Routing 14
    2.8 Message life cycle 14
    2.9 Routing log 15
    2.10 Orphans 15
    2.11 Protocol conversion 15
        2.11.1 CIMD2 (Nokia) 16
        2.11.2 SMPP (SMS Forum) 16
        2.11.3 UCP/EMI (CMG) 16
        2.11.4 OIS (Sema) 16
    2.12 Performance 16
    2.13 Support 17
3. Installing or upgrading EMG 18
    3.1 Before installing EMG 18
    3.2 Download software 18
    3.3 Get license key 18
    3.4 Install software 18
        3.4.1 Full distribution 18
        3.4.2 Binaries-only 20
    3.5 Configure software 21
    3.6 Starting, stopping and refreshing the server 21
    3.7 Persistence 21
4. Configuration 23
    4.1 General 23
    4.2 Refreshing the server 23
    4.3 Testing a configuration 23
5. Connectors 25
    5.1 Connector types 25
    5.2 Connector modes 25
    5.3 Static vs non-static 25
    5.4 Connector states 26
    5.5 Instances 26
    5.6 Message types 27
    5.7 Mappings 27
    5.8 Address rewriting 28
        5.8.1 Masquerading 29

5.8.2 Source Address Translation (SAT) 30
5.9 Inheritance and virtual connectors 31
5.10 Limiting connector queue sizes 32
5.11 Sample configurations 32
5.11.1 Incoming MGP supporting up to 3 connections 32
5.11.2 Incoming SMPP supporting up to 10 connections 32
5.11.3 Incoming CIMD2 supporting 1 connection 32
5.11.4 Outgoing CIMD2 33
5.11.5 Outgoing UCP using authentication via operation 60 33
5.11.6 Outgoing UCP via modem 33
5.11.7 Outgoing HTTP 34
5.11.8 Outgoing EBE 34
5.11.9 Outgoing GSM 34
5.11.10 Outgoing OIS 35
6. Routing 36
6.1 Specifying a routing criteria for a message 36
6.2 Specifying a routing criteria for a connector 36
6.2.1 The REDIRECT keyword 36
6.3 Routing table 37
6.4 Failover 38
6.5 Load balancing 38
6.6 User-based routing 38
6.7 Keyword-based routing 39
6.7.1 Sample scenario 39
6.7.2 Setting up the connectors 39
6.7.3 Setting up the routing table 40
6.7.4 Keyword sessions 40
6.8 Concatenated Message Routing (CMR) 41
7. Sending messages 42
7.1 Using HTTP connector 42
7.2 Using SMTP connector 42
7.2.1 Sample outgoing SMTP session 43
7.3 Using EBE connector 43
7.4 Using emgsend 44
7.5 Using emgclient 45
7.6 Sending MMS 45
8. Receiving messages 47
8.1 Using HTTP connector 47
8.2 Using SMTP connector 47
8.2.1 Sample incoming SMTP session 48
8.2.2 Preventing SMTP relaying 49
8.3 Receiving WAP push via PAP 49
9. Delivery receipts (DLR) 51
9.1 Overview 51
9.2 Requesting a DLR 51
9.2.1 Using emgsend 51
9.2.2 Using connector keyword 51
9.3 Routing DLRs 52
9.4 DLRs and buffered status 52
10. Logging 53

10.1 Location of log files 53
10.2 Format of log files 54
10.2.1 Connector log file 54
10.2.2 Sample incoming connector log file 55
10.2.3 Sample outgoing connector log file 55
10.2.4 PDU log files 55
10.3 Log file rotation 56
10.3.1 Log file rotation based on time 56
10.3.2 Log file rotation based on size 56
10.4 Logging to a database 56
11. Security 57
11.1 Access control 57
11.2 Authentication 57
11.3 Blacklists and whitelists 57
11.4 Using SSL 58
11.4.1 Outgoing SSL without a certificate 58
11.4.2 Incoming SSL 58
12. Database support 59
12.1 Getting started with a DB 59
12.1.1 Installing the database server 59
12.1.2 Initializing the database 59
12.1.3 Creating a database profile 59
12.1.4 Referencing a database profile 60
12.2 Using the database 60
12.2.1 Putting the message route log in a database 60
12.2.2 Putting the connector log in a database 60
12.2.3 User authentication from database 60
12.3 Schema version handling 61
13. Performance 62
13.1 Hardware and operating system 62
13.1.1 CPU 62
13.1.2 RAM 62
13.1.3 Disk 62
13.1.4 Operating system 63
13.2 Protocols 63
13.3 Instances 63
13.4 Other issues 63
13.4.1 Modifying message content 63
13.4.2 Server logging and debug mode 64
13.5 About benchmarks 64
A. Command reference 65
A.1 emgclient 65
A.2 emgd 66
A.3 emgsend 67
A.4 emgstat 68
A.5 mmscomp 69
A.6 pushtohex 70
A.7 rttltohex 71
B. Configuration options 72
B.1 General options 72

B.1.1 BLACKLIST 72
B.1.2 CMREXPIRE 72
B.1.3 CONNECTORLOGDB 72
B.1.4 CONNECTOR_LOGLEVEL 72
B.1.5 DBPROFILE 73
B.1.6 DISABLE_CREDITS 73
B.1.7 DLRSSIZE 73
B.1.8 DLRVP 73
B.1.9 DNSTHREADS 73
B.1.10 LOGFLUSHSIZE 73
B.1.11 LOGLEVEL 74
B.1.12 LOGYEAR 74
B.1.13 MAXTOTALQUEUESIZE 74
B.1.14 NOEXPIRE 74
B.1.15 NOFLUSH 75
B.1.16 NOLOGSERVER 75
B.1.17 ORPHANSSIZE 75
B.1.18 PERSISTFILES 75
B.1.19 PERSISTSIZE 75
B.1.20 ROTATELOGS 76
B.1.21 ROUTEDLR 76
B.1.22 ROUTELOGDB 76
B.1.23 ROUTELOGSIZE 76
B.1.24 ROUTING 77
B.1.25 SERVERNAME 78
B.1.26 SHMKEY 78
B.1.27 SPOOLDIR 78
B.1.28 SSL_KEYFILE 78
B.1.29 SSL_PASSWORD 78
B.1.30 TABLE_PREFIX 79
B.1.31 TIME_OFFSET 79
B.1.32 WHITELIST 79
B.2 Connector options 79
B.2.1 ACCESS 79
B.2.2 ADDRESS 80
B.2.3 ADDRESSRANGE 80
B.2.4 ALLOWROUTE 80
B.2.5 AUTHCODE 80
B.2.6 AUTHNPI 80
B.2.7 AUTHTON 80
B.2.8 AUTOMATICTONNPI 81
B.2.9 AUTOMATICTONNPI_ALPHANUMERIC_NPI 81
B.2.10 AUTOMATICTONNPI_ALPHANUMERIC_TON 81
B.2.11 AUTOMATICTONNPI_DEFAULT_NPI 81
B.2.12 AUTOMATICTONNPI_DEFAULT_TON 81
B.2.13 AUTOMATICTONNPI_SHORTCODE_NPI 81
B.2.14 AUTOMATICTONNPI_SHORTCODE_TON 81
B.2.15 BLACKLIST 81
B.2.16 CDMA 82
B.2.17 CDRFIELDS 82

B.2.18 CMR 82
B.2.19 DEFAULT_CHARCODE 82
B.2.20 DEFAULT_DESTADDRNPI 82
B.2.21 DEFAULT_DESTADDRNPI_IN 82
B.2.22 DEFAULT_DESTADDRTON 82
B.2.23 DEFAULT_DESTADDRTON_IN 82
B.2.24 DEFAULT_DLR 82
B.2.25 DEFAULT_DLRADDRESS 83
B.2.26 DEFAULT_MSGTYPE 83
B.2.27 DEFAULT_PROTOCOLID 83
B.2.28 DEFAULT_QPRIORITY 83
B.2.29 DEFAULT_SMSCOP 83
B.2.30 DEFAULT_SOURCEADDR 83
B.2.31 DEFAULT_SOURCEADDRNPI 84
B.2.32 DEFAULT_SOURCEADDRNPI_IN 84
B.2.33 DEFAULT_SOURCEADDRTON 84
B.2.34 DEFAULT_SOURCEADDRTON_IN 84
B.2.35 DEFAULT_VP 84
B.2.36 DELAYFIRSTMESSAGE 84
B.2.37 DESTFULLNAME 84
B.2.38 DLR_ERR_HEX 84
B.2.39 DLREXPIRES 85
B.2.40 DLR_EXPIRES_STATUS 85
B.2.41 DLRIGNOREKEYWORD 85
B.2.42 DLRMINMATCHLENGTH 85
B.2.43 DOMAIN 85
B.2.44 FAILOVER 85
B.2.45 FIRST_TRN 86
B.2.46 FORCE_CHARCODE 86
B.2.47 FORCECLOSE 86
B.2.48 FORCE_DCS 86
B.2.49 FORCE_DESTADDR 86
B.2.50 FORCE_DESTADDR_IN 86
B.2.51 FORCE_DESTADDRNPI 86
B.2.52 FORCE_DESTADDRNPI_IN 87
B.2.53 FORCE_DESTADDRTON 87
B.2.54 FORCE_DESTADDRTON_IN 87
B.2.55 FORCE_DESTPORT_IN 87
B.2.56 FORCE_DLR 87
B.2.57 FORCE_MESSAGE 87
B.2.58 FORCE_PRIORITY 87
B.2.59 FORCE_PROTOCOLID 87
B.2.60 FORCE_SERVICETYPE 87
B.2.61 FORCE_SERVICETYPE_IN 88
B.2.62 FORCE_SOURCEADDR 88
B.2.63 FORCE_SOURCEADDR_IN 88
B.2.64 FORCE_SOURCEADDRNPI 88
B.2.65 FORCE_SOURCEADDRNPI_IN 88
B.2.66 FORCE_SOURCEADDRTON 88
B.2.67 FORCE_SOURCEADDRTON_IN 88

B.2.68 FORCE_SOURCEPORT_IN 88
B.2.69 FORCE_VP 89
B.2.70 GSMNOSCA 89
B.2.71 GSMSTORE 89
B.2.72 HEXID 89
B.2.73 HOME_IMSI 89
B.2.74 HOME_VLR 89
B.2.75 IDLETIMEOUT 90
B.2.76 IGNOREMAXTOTALQUEUESIZE 90
B.2.77 INHERIT 90
B.2.78 INITSTRING 90
B.2.79 INSTANCES 90
B.2.80 INTERFACEVERSION 91
B.2.81 KEEPALIVE 91
B.2.82 LIBRARY 91
B.2.83 LOCALDOMAINS 91
B.2.84 LOCALIPS 91
B.2.85 LOGLEVEL 92
B.2.86 LOGMESSAGE 92
B.2.87 LOGPDU 92
B.2.88 LONGMESSAGE 92
B.2.89 LONGMODE 92
B.2.90 MAPPING 92
B.2.91 MASQUERADE 93
B.2.92 MAXFAILEDCONNECTS 93
B.2.93 MAXFAILEDSLEEP 93
B.2.94 MAXMESSAGELENGTH 93
B.2.95 MAXTRIES 93
B.2.96 MESSAGELENGTH 93
B.2.97 MESSAGEMODE 94
B.2.98 MESSAGES_PER_REQUEST 94
B.2.99 MIMEBOUNDARY 94
B.2.100 MMS_TEXT_CHARSET 94
B.2.101 MODE 94
B.2.102 MODEM 94
B.2.103 MSGDELAY 95
B.2.104 MSGRETRYTIME 95
B.2.105 MTBILLING 95
B.2.106 NOBINARYMAPPING 95
B.2.107 NOUCS2MAPPING 95
B.2.108 NOUSERMESSAGEREFERENCE 95
B.2.109 OPSENTEXPIRES 95
B.2.110 OPS_MAXINCOMING 95
B.2.111 OPS_MAXOUTGOING 95
B.2.112 OPS_MAXOUTSTANDING 96
B.2.113 OPS_MAXPERSESSION 96
B.2.114 ORIGIN 96
B.2.115 PARSEMESSAGE 96
B.2.116 PASSWORD 96
B.2.117 PLUGIN 97

B.2.118 POLLRECEIVE 97
B.2.119 PREFIX 97
B.2.120 PRESERVESAR 97
B.2.121 PROMPT 97
B.2.122 PROTOCOL 97
B.2.123 QUOTEDREPLY_SEPARATOR 97
B.2.124 QUOTEDSUBJECT 98
B.2.125 REDIRECT 98
B.2.126 REGEXP_DESTADDR 98
B.2.127 REGEXP_DESTADDR_IN 98
B.2.128 REGEXP_KEYWORD 98
B.2.129 REGEXP_MESSAGE 98
B.2.130 REGEXP_SOURCEADDR 98
B.2.131 REGEXP_SOURCEADDR_IN 99
B.2.132 REJECT_EMPTY 99
B.2.133 REMOVEPREFIX 99
B.2.134 REMOVEPREFIX_SOURCEADDR 99
B.2.135 REPLACEPREFIX 99
B.2.136 REPLACEPREFIX_IN 99
B.2.137 REPLACEPREFIX_SOURCEADDR 99
B.2.138 REPLACEPREFIX_SOURCEADDR_IN 100
B.2.139 REQUIREPREFIX 100
B.2.140 REQUIREPREFIX_SOURCEADDR 100
B.2.141 RETRYSCHEME 100
B.2.142 RETRYTIME 100
B.2.143 REVDLR 100
B.2.144 REVDLR_IN 100
B.2.145 ROUTE 101
B.2.146 ROUTEDLR 101
B.2.147 ROUTING 101
B.2.148 SATPOOL_CREATE 101
B.2.149 SATPOOL_CREATE_IN 101
B.2.150 SATPOOL_LOOKUP 101
B.2.151 SATPOOL_LOOKUP_IN 101
B.2.152 SCAADDR 102
B.2.153 SCAADDRNPI 102
B.2.154 SCAADDRTON 102
B.2.155 SENDERADDRESS 102
B.2.156 SEPARATOR 102
B.2.157 SERVICETYPE 102
B.2.158 SIMULATE 103
B.2.159 SMPPTZ 103
B.2.160 SOURCEADDR_GSM 103
B.2.161 SOURCEFULLNAME 103
B.2.162 SSL 103
B.2.163 SSL_KEYFILE 103
B.2.164 SSL_PASSWORD 103
B.2.165 STATIC 104
B.2.166 SUBADDRESS 104
B.2.167 SUBJECT 104

B.2.168 SUPPRESS_EMGHEADERS 104
B.2.169 SYSTEMTYPE 104
B.2.170 TCPSOURCEIP 104
B.2.171 TCPSOURCEPORT 105
B.2.172 THROUGHPUT 105
B.2.173 TYPE 105
B.2.174 UDHVIAOPTIONAL 105
B.2.175 USEDELTIME 105
B.2.176 USEPRIORITY 106
B.2.177 USERDB 106
B.2.178 USERDB_FORCE 106
B.2.179 USERNAME 106
B.2.180 USERS 106
B.2.181 USESENDER 107
B.2.182 USESUBJECT 107
B.2.183 VASID 108
B.2.184 VASPID 108
B.2.185 VIRTUAL 108
B.2.186 WAITBEFORECONNECT 108
B.2.187 WAITDELAY 108
B.2.188 WAITFOR 108
B.2.189 WHITELIST 108
B.2.190 XAUTH 109
B.2.191 XAUTHPASSWORD 109
B.2.192 XAUTHUSERNAME 109
B.2.193 XPASSWORD 109
B.2.194 XUSERNAME 109
B.3 DB options 109
B.3.1 DBNAME 110
B.3.2 HOST 110
B.3.3 INSTANCES 110
B.3.4 PASSWORD 110
B.3.5 PORT 110
B.3.6 TYPE 110
B.3.7 USERNAME 110
B.4 SAT pool options 111
B.4.1 ADDRESSRANGE 111
B.4.2 EXPIRE 111
B.4.3 QUOTEDREPLY 111
B.4.4 THREADED 111
B.5 Domain options 111
B.5.1 MAILSPERMINUTE 112
B.5.2 MAILSPERSESSION 112
B.5.3 PORT 112
B.5.4 RETRYTIME 112
B.5.5 SESSIONS 112
B.6 Plugin options 112
B.6.1 CONFIG 113
B.6.2 INSTANCES 113
B.6.3 LIBRARY 113

B.6.4 OFFSET 113
C. MGP options 114
C.1 Option keys in numeric order 115
C.2 Options reference 116
C.2.1 BlLLINGID (79) 116
C.2.2 CHARCODE (28) 116
C.2.3 CONCATSMSMAX (75) 116
C.2.4 CONCATSMSREF (73) 116
C.2.5 CONCATSMSSEQ (74) 116
C.2.6 CONNECTOR (59) 117
C.2.7 DCS (113) 117
C.2.8 DELTIME (20) 117
C.2.9 DESTADDR (8) 117
C.2.10 DESTADDRNPI (10) 117
C.2.11 DESTADDRTON (9) 117
C.2.12 DESTPORT (12) 118
C.2.13 DLR (19) 118
C.2.14 DLRID (81) 118
C.2.15 ENDMSECS (96) 118
C.2.16 ENDSECS (95) 118
C.2.17 HPLMNADDR (110) 118
C.2.18 ID (1) 118
C.2.19 LRADDR (108) 118
C.2.20 LRPID (109) 118
C.2.21 MESSAGE (16) 118
C.2.22 MESSAGELEN (17) 119
C.2.23 MSGCLASS (27) 119
C.2.24 MSGTYPE (15) 119
C.2.25 NOTE (97) 119
C.2.26 OTOA (112) 119
C.2.27 OUTCONNECTOR (60) 119
C.2.28 PRIORITY (32) 119
C.2.29 PROTOCOLID (43) 120
C.2.30 QPRIORITY (118) 120
C.2.31 REMOTEIP (34) 120
C.2.32 REPLACEPID (107) 121
C.2.33 REPLYPATH (31) 121
C.2.34 ROUTE (38) 121
C.2.35 SCTS (21) 121
C.2.36 SMSCID (64) 121
C.2.37 SOURCEADDR (2) 121
C.2.38 SOURCEADDRNPI (4) 121
C.2.39 SOURCEADDRTON (3) 121
C.2.40 SOURCEPORT (6) 121
C.2.41 STARTMSECS (94) 121
C.2.42 STARTSECS (93) 122
C.2.43 STATUS (61) 122
C.2.44 SUBJECT (111) 122
C.2.45 UDH (14) 122
C.2.46 UDHI (106) 122

C.2.47 USER (30) 122
C.2.48 VP (18) 123
C.2.49 XUSERNAME (119) 123
D. Error codes 124
D.1 CIMD2 124
D.2 SMPP 124
D.3 UCP/EMI 125
D.4 OIS 125
D.5 HTTP 125
D.6 SMTP 126
E. SMSC inter-connectivity checklist 127
E.1 Your requirements 127
E.1.1 Send messages 127
E.1.2 Receive messages 127
E.1.3 Type of messages 127
E.1.4 Performance or message volume 127
E.1.5 Support and service 127
E.2 SMSC connection 127
E.2.1 Type of connection 127
E.2.2 Protocol 128
E.2.3 Performance 128
E.2.4 Security 128
E.3 Getting started 128
E.3.1 Account information 128
E.3.2 Sending the first message 129
E.3.3 Receiving a message 129
F. Change history 130
F.1 EMG 3.0.16.14262 130
F.2 EMG 3.0.15.12050 130
F.3 EMG 3.0.14.11726 131
F.4 EMG 3.0.13.10913 131
F.5 EMG 3.0.12.10307 131
F.6 EMG 3.0.11.9982 132
F.7 EMG 3.0.10.9505 132
F.8 EMG 3.0.9.9284 133
F.9 EMG 3.0.8.9115 133
F.10 EMG 3.0.7.8470 134
F.11 EMG 3.0.6.8403 134
F.12 EMG 3.0.5.7719 135
F.13 EMG 3.0.4.6983 136
F.14 EMG 3.0.3.6691 137
F.15 EMG 3.0 137
Index 138

1. Acknowledgements

1.1 OpenSSL

Acknowledgements in accordance with the OpenSSL license:

- This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)
- This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org)

1.2 LibXML

Acknowledgements in accordance with the LibXML license:

Copyright (C) 1998-2003 Daniel Veillard. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

1.3 PCRE

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England.

1.4 PCRS

Perl regular expression support is provided by the PCRS library package

Written and Copyright (C) 2000, 2001 by Andreas S. Oesterhelt, .andreas@oesterhelt.org.

2. Overview

Enterprise Messaging Gateway (EMG) is a messaging platform that can act as a SMS/MMS gateway, protocol converter or some other kind of mediation gateway.

Usually the job of EMG is to relay messages, performing message conversion and translation depending on the connectors used to receive and send the message.

2.1 Licensing

Enterprise Messaging Gateway licensing is based on number of messages per second per server. More specifically it is the total number of messages that all incoming connectors, server-wide, will accept per second. If, for example, there are two incoming connectors in a system licensed for 30 messages per second and they are fed with 20 messages per second each, the EMG server will impose a small delay so that in practice the connectors will process on average 15 messages per second each.

EMG is also licensed per server so that one license permits installation of EMG on one specific server. The license is generated for a specific server identity (hostid). For Solaris systems this equals to the output from the command hostid and for Linux it is the MAC, or hardware, address for the first network adapter in the system, interface eth0.

There is no license-related limitation as to how many connectors can be defined or how many clients or SMSC connections an EMG server can handle. One server can handle multiple client and SMSC connections using different protocols.

In order to be able to move a license from one server to another a new license needs to be generated by Nordic Messaging Technologies AB or one of its representatives.

2.2 Messages

The message is the most important entity in EMG. A message is represented as a unique object, identified by its unique message id, that is assigned a number of properties of which some can be set by the user and some are set by the system. Each property is a key-value pair, where the numeric key identifies the property. Usually a message are assigned at least the following, user-level, properties:

- ID, unique message id (key value 1)
- DESTADDRESS, destination address or recipient (8)
- MESSAGE, message data (16)

These properties are referred to as MGP options, where MGP stands for the Messaging Gateway Protocol. A list of key values can be found in the chapter “MGP options”.

2.3 Binary messages and User-Data Header (UDH)

EMG supports binary, or 8-bit, messages including User-Data Header (UDH). This enables sending of ringtones, logos, WAP push, OTA and other messages containing other information than plain text. When converting from one protocol to another message and optionally splitting messages, UDH properties are preserved.

2.4 Long messages

According to the GSM specification one message (SMS) cannot exceed 160 septets or 140 octets in length. However, sometimes it is necessary to send more information than that in one message. This is solved by splitting the message into multiple messages and indicate that they belong together by using the GSM 3.40 UDH feature “concatenated message”. EMG handles both splitting of messages and setting the corresponding UDH option automatically. In EMG one message is translated into one or more Physical Data Units (PDUs). Each PDU corresponds to one transferred SMS.

2.5 MMS

Starting with EMG 3, EMG also supports sending and receiving multimedia messages, MMS. Protocols such as MM7, EAIF and PAP are supported. Basic convserion functionality is also included, for example e-mail to MMS and MMS to e-mail conversion.

2.6 Connectors

Messages are sent and received through so called connectors in the server. Each connector is of type incoming and outgoing, exists in one or more instances and implement one of the supported protocols. Outgoing connectors support failover and load balancing via the routing table.

2.7 Routing

The process of passing a message from one connector to another in order for it to reach its final destination is referred to as routing. The routing decision is based on the information in the routing table and message and connector properties.

2.8 Message life cycle

A message that is relayed through EMG passes a number of stages:

- Message is received via a connector and its attached parameters are parsed.
- A routing decision is made dependant on message information, connector information and the routing table. An outgoing connector is selected.
- Message is placed in the queue for the selected connector OR if an outgoing connector could not be selected the message is considered an orphan.

- Message is delivered through the connector if possible. If the connector fails, failover options are checked and the message may be re-routed correspondingly to another connector.

When a message is deleted or queried, first the routing log is checked. If the message has been relayed (usually to an SMSC) the endpoint is queried if possible. Message status RELAYED indicates that the message has been relayed but the final result is unknown. However, status DELIVERED indicates that the message has been delivered to the recipient. When a message is sent to an SMSC the message status is RELAYED until further information is available while if for example a HTTP connector delivers the message the status will be DELIVERED.

Messages placed in connector queues, or orphaned, can be made to expire when the validity period, if present for the message, is reached.

When running without message persistence option messages are stored in RAM during their life cycle and will be lost if server is stopped ungracefully. With the message persistence option messages, DLRs and keyword sessions are persisted on file for their life time.

2.9 Routing log

When a message is processed by EMG it is entered into the routing log. The routing log is used to keep track of the most recent messages, their state and what connector they were received and sent out at. The maximum number of entries in the routing log is defined by the keyword ROUTELOGSIZE. When the log exceeds the maximum size the oldest entry will be discarded.

2.10 Orphans

When an incoming message cannot be routed or is routed to a connector that does not exist it will be added to a special queue for so called orphaned messages. The maximum size of this queue is defined by the keyword ORPHANSSIZE. When the queue exceeds the maximum size the oldest entry will be discarded. If the value for this keyword is set to 0 orphaned messages will be discarded directly.

When the server is refreshed the queue for orphaned messages will be processed and if, for example, the routing table has changed orphaned messages may be successfully re-routed to a destination connector.

2.11 Protocol conversion

The different protocols supported by EMG differs somewhat in functionality and possibilities. This is especially true when going from an older version of a protocol to a newer version. This implies that some kind of conversion must take place. Some parameters may need to be added, some converted and some maybe even removed.

2.11.1 CIMD2 (Nokia)

CIMD2 supports two different types of Message Centers, SMSC and USSD. Some options are specific for one of the message center types.

The SMSC distinguishes between three types of SME:s (Short Message Entities, in this case the EMG application):

1 Send-only, the SME can only send messages
2 Querying, the SME requests delivery of messages or polls for messages
3 Receiving, the SME is always ready to receive a message. The querying type setup seems to be quite common and in this case the POLLRECEIVE keyword needs to be specified so that the SMSC will deliver messages.

EMG supports CIMD2 2-0en.

2.11.2 SMPP (SMS Forum)

This protocol, in version 3.4, is designed to support a variety of different messaging centers in different network types and hence is the protocol that provides the largest amount of options. However, for SMS in a GSM network many of the options are not used. SMPP has evolved to be the number one messaging standard and is getting more and more widely used.

EMG can be configured to use both the submit_sm and the data_sm SMPP operations depending on protocol version (specified by INTERFACEVERSION) and the value of DEFAULT_SMSCOP. The submit_multi_sm operation is currently not supported.

SMPP 3.4 is a much more thorough specification than SMPP 3.3.

EMG supports SMPP 3.3 and SMPP 3.4.
It is implemented by the Logica SMSC and others.

2.11.3 UCP/EMI (CMG)

There a number of different operations for sending a message that remain from earlier versions of the protocol but which are now obsolete. The default operation and the operation that should be used is the Submit Short Message (51) operation.

EMG supports UCP/EMI 3.5 and UCP/EMI 4.0.

2.11.4 OIS (Sema)

OIS is supported over TPC/IP. No authentication is used for these connections but full support for text, binary and messages using UDH is available.

EMG supports OIS according to the version 5.8 specification.

2.12 Performance

EMG is licensed based on messages per second. Above 200 messages per second the license is unlimited. The achieved performance depend on a number of fac-

tors: Hardware, operating system, protocol used, clients, SMSC etc. However, connectors using the messaging protocols supported (CIMD2, SMPP and UCP) are capable to process more than 4.000 messages per second in an ideal environment.

In benchmarks where we short-circuit SMPP connectors (that is we use one server which provides both the outgoing and incoming connectors) and a message is looped, we achieve 200+ messages per second on a low-end Sun Sparcstation 5 with 256 MB RAM running Solaris 8. This indicates that EMG is capable of very high throughput on relatively basic hardware.

Moreover, EMG on Linux generally outperforms EMG on Solaris probably due to the fact that the TCP/IP implementation on Linux is very efficient. On the other hand we prefer Solaris as a stable and reliable platform for business-critical applications.

2.13 Support

Support is available from your reseller and requires a valid support agreement. If you are missing information about your reseller, please contact Nordic Messaging Technologies via e-mail: support@nordicmessaging.se.

Make sure you include a detailed problem description, your license or serial number and detailed contact information. If you want us to call you make sure you indicate your geographical location and time zone.

Nordic Messaging Technologies is located in Stockholm, Sweden (CET).

3. Installing or upgrading EMG

EMG installation/upgrade process includes several steps:

- Download software
- Get license key
- Install software
- Configure software
- Start server

3.1 Before installing EMG

We recommend starting out by applying relevant patches to your operating system. This particularly applies to Sun Solaris. We recommend applying Sun's most current Recommended Patch Clusters for your Solaris version. Sun patches are available from http://sunsolve.sun.com.

To list installed patches in Solaris use "showrev -p".

3.2 Download software

Software can be downloaded from http://www.nordicmessaging.se/download.

You can choose either to download a full distribution or a binaries-only distribution. When installing EMG for the first time you need to download the full distribution. However, when upgrading the binaries-only distribution is enough unless you want new versions of the configuration files. New configuration files are not necessary since EMG is backwards compatible with earlier releases.

3.3 Get license key

Contact your distributor or Nordic Messaging Technologies in order to get your evaluation license key or to purchase a permanent license.

When upgrading from earlier versions of EMG (EMG 1, 2 or 2.5) you need a new license key. If there is an active support agreement in place for your license you can get the new license key by sending e-mail support@infoflexconnect.se requesting the new key. Include the current license key in the e-mail. Delivery time for licenses may be up to two workdays.

3.4 Install software

3.4.1 Full distribution

Extract the components under the root directory. This will create a directory, /tmp/emg2, change to the new directory and execute the install script. The install script will ask a number of questions regarding your configuration and prompt you for the the license key.

```
# cd /
# gzip -d emg30-solaris-full.tar.gz
# tar xvf emg30-solaris-full.tar
# cd /tmp/emg
# sh ./INSTALL
```

During the installation you will be able to choose a directory for the configuration files. By default this directory is /etc/emg and will be referred to as EMGDIR in this documentation. If you choose to install the software using another directory for the configuration files, the environment variable EMGDIR needs to be set correspondingly in order for the programs to find the configuration files.

Example for setting the EMGDIR variable to an alternate directory, /opt/emg/etc, in Bourne shell or equivalent:

```
# export EMGDIR
# EMGDIR=/opt/emg/etc
```

After running the installation you should be able to execute "emgd -v" in order to display your license information.

Sample output from an INSTALL script session on Solaris:

```
# sh ./INSTALL

* * *  Enterprise Messaging Gateway 2.5 - INSTALLATION  * * *

EMG can be owned by the user root or by another user.
What user should be the owner of the EMG files? [root]
What group should be the group for the EMG files? [root]
Where should Enterprise Messaging Gateway configuration
configuration files go? [/etc/emg]
Where should EMG executables go? [/usr/bin]

* * *  Make sure this directory is in your PATH!  * * *
Creating directories...OK
Moving programs...OK
Moving data files...OK
Creating configuration file...OK
LICENSE INFORMATION
Enter your license information EXACTLY as received.
HOSTID     :
COMPANY    : DEMO
TELNO      : 123456
SERIAL     : emgDEMO
LICENSEDATA: 30/Aug/2003-0-0-1
ACTIVATION : I-6l5H94FafYn4A6

By creating /etc/init.d/emg the Enterprise Messaging Gateway
server
can be automatically started on system boot.
Should the server be automatically started on system boot? [y]

Creating /etc/init.d/emg.
Linking /etc/init.d/emg to /etc/rc2.d/S99emg.

INSTALLATION FINISHED!
```

```
======================

The configuration file is placed in /etc/emg and named server.cfg.
The EMG server, emgd, can be started with "emgd".

Check the file /etc/emg/README for more information.
```

3.4.2 Binaries-only

The easiest way to upgrade an existing EMG software installation is to use the binaries-only distribution. Since EMG is backwards compatible with earlier versions no configuration changes should be needed unless new functionality needs to be accessed.

The upgrade procedure is as follows:

Log on to your system as the system administrator or the EMG user.

Find your current EMG binaries, for example emgd. They are usually located in /usr/bin. You can also execute "type emgd", which will locate the binary if it is in your current path.

Download the patch release tar-archive, in this example we use emg25-solaris-binaries.tar.gz, and save it in the file system under /tmp or similar.

Uncompress the archive:

```
gzip -d /tmp/emg30-solaris-binaries.tar.gz
```

Stop emgd, "emgd -stop", and then make sure there are no emgd processes running, "ps -ef | grep emgd". All emgd processes need to be stopped before replacing the binaries.

Backup your current binaries in order to be able to revert to these files if needed. Sample backup procedure:

```
mkdir /tmp/emg-backup.020501
cp /usr/bin/emg* /tmp/emg-backup.041030
```

Extract the new binaries:

```
cd /usr/bin
tar xvf /tmp/emg30-solaris-binaries.tar
```

Make sure they have appropriate owner and permissions (we use root in this example):

```
chown root /usr/bin/emg*
chmod 555 /usr/bin/emg*
chmod 500 /usr/bin/emgd
```

Verify that the new binaries execute ok (the command below should not give any error messages):

```
emgd -help
```

Verify that the server configuration is ok (if not, make required adjustments):

```
emgd -verify
```

Start the server:

```
emgd
```

Verify that the server is running:

```
ps -ef|grep emgd
```

You should now be up and running with your upgraded software.

3.5 Configure software

Configuring the software usually involves

- Creating a server.cfg that includes configuration for the needed connectors.
- Creating a client.cfg with hostname, port and possibly authentication information for MGP clients.

After the installation there will be a sample configuration in EMGDIR.

3.6 Starting, stopping and refreshing the server

The server can be started by simply running

```
# emgd
```

It is possible to set the debug level to DEBUG using the "-debug" option. It will also send the debug output to stdout.

```
# emgd -debug
```

The server can be stopped using

```
# emgd -stop
```

To refresh the server use

```
# emgd -refresh
```

There are 3 different ways to refresh the server:

1 Run "emgd -refresh"
2 Run "kill -1 <pid>" where pid is the parent emgd process
3 Use emgclient or another MGP client (requires MGP administrator privilege)

EMG 3 also supports reloading which is similar to refresh with the difference that open sessions are not closed as is the case during a refresh.

Reloading the server is done by running “emgd -reload”.

3.7 Persistence

When the server is stopped the connector messages queues and the routing log are flushed to disk. The entries will be stored in the files queues.dat and route-

log.dat respectively. The format of these files is one MGP option key-value pair per row with an extra empty line as message separator.

If message persistence is enabled in the license flush files are not used. Instead all messages, DLRs and SAT entries are stored in the file system immediately upon reception and for the whole message life cycle. When the message has reached its final status (delivered or failed) it is also removed from disk. This way no messages will ever be lost even if the server terminates unexpectedly or crashes.

4. Configuration

4.1 General

Software configuration is done by editing the configuration files. For server configuration the file EMGDIR/server.cfg is used and for client configuration the file EMGDIR/client.cfg is used. EMGDIR defaults to /etc/emg as described in the Installation chapter.

The configuration files contain a number of keywords used to control the behavior of EMG.

In order to use EMG at least two connectors need to be defined in the server.cfg file. However, it is most common that at least three connectors are defined:

- One incoming connector for MGP (used by the EMG client programs)
- One incoming connector where messages will be received
- One outgoing connector which connects to an SMSC or similar

For more information on connectors and how to configure them please consult the chapter Connectors.

4.2 Refreshing the server

After making changes to the server configuration file the changes will not be in effect until the server is either restarted, refreshed or reloaded. The routing table, user files and most options in the configuration file will be refreshed with the following exceptions:

- Connectors cannot be deleted or added
- The number of connector instances cannot be changed
- The connector protocol cannot be changed

If any of these options need to be changed you would need to stop and start the server.

During a refresh all open connections will be closed gracefully while during a reload connections will not be close but only temporarily suspended.

4.3 Testing a configuration

After installing and configuring EMG consider the following:

- When starting emgd does any messages that indicate a problem with the configuration file show up in the general log file (default EMGDIR/log/general)?
- If EMG is to be used as a converter with both incoming and outgoing connectors, first verify that the outgoing connector(s) work(s). Use the emgsend command-line utility to “inject” messages into EMG and use the server.cfg keyword ROUTE or the routing table to make sure incoming messages from emgsend (MGP protocol) will be routed to the correct outgoing connector.

- Check the connector logs (default EMGDIR/connector.*). For each message received and sent on a connector there will be a corresponding entry in the connector log file.
- Use the “-debug” option with emgd to generate complete debug information. This information will be appreciated by NMT support engineers when helping out in troubleshooting an installation. The debug information will both be displayed on stdout and stored in the general log file. You may need to empty the log file from old information by simply removing the file before starting “emgd -debug”. The file will be created if it does not exist when emgd is started.

5. Connectors

Connectors provide the interface between Enterprise Messaging Gateway and other systems and applications. For example, one connector can connect to an operator's SMSC and another connector can accept incoming messages via HTTP. Each connector can be available in one or more instances. The core functionality of EMG is to route messages from one connector to another.

When testing EMG or running benchmarks it can be useful to route message from an outgoing EMG connector to an incoming connector in the same server. We call this short-circuiting connectors.

5.1 Connector types

A connector can be one of two types: Incoming or outgoing. An incoming connector accepts incoming connections and an outgoing connector initiates connections. Please note however that whether a connector is incoming or outgoing does not tell anything about the direction of flow of the messages on the connector. Messages can be received on an outgoing connection, for example when connecting to a Nokia SMSC.

Both incoming and outgoing connectors can exist in 0 or more instances. However, normally an outgoing connector only needs one instance while an incoming connector needs several instances in order to be able to serve several simultaneous connections.

5.2 Connector modes

Connectors can be defined to be transmit-only, receive-only or both transmit and receive. This is not the same as incoming vs outgoing. For example when connecting to a Nokia SMSC an outgoing connector can poll for messages, which will then be received.

The connector mode is specified using the MODE keyword. By default connectors can both transmit and receive messages. Incoming messages on a transmit-only connector will be rejected.

In SMPP 3.3 however sessions will be used either for transmit or receive depending on the bind operation used. This means that in order to be able to both send and receive messages at least two connectors must be defined.

5.3 Static vs non-static

An outgoing connector can be defined as static using the STATIC keyword. This indicates that the connector should connect to the remote entity immediately even if there are no messages to send. The connector will still respect any idle timeout specified to if IDLETIMEOUT is set to a non-zero value the connector will disconnect after being idle the specified number of seconds and the immediately reconnect since it is configured as static.

Usually you would use IDLETIMEOUT=0 and set KEEPALIVE to a value, for example 60 (seconds). This will keep the connection open by sending keepalive operations periodically to ensure the connection up.

5.4 Connector states

A connector can be in exactly one of the following states at a specific time:

IDLE
: Connector is just created or has been dynamically disconnected. Static connectors should never be in this state during normal operation.

CONNECTED
: Connector is connected to remote endpoint but has not been authenticated.

BOUND
: Connector is authenticated and allowed to transmit and receive messages.

TERMINATING
: Connector will perform a graceful shutdown as soon as possible.

DEAD
: Connector is shut down.

ERROR
: Connector has failed a repeated number of times and is awaiting a new retry period.

On top of the above a connector can be put “on hold”, which means messages are accepted in the queue but no messages are transmitted. A message can be put on hold using the emgclient utility or another implementation using the MGP protocol.

5.5 Instances

A connector is available in 0 or more instances. In order for the connector to be active and be able to send or receive messages it must be available in at least one instance.

Outgoing connectors usually only needs to be available in one instance. From EMGs point of view one instance is enough to handle hundreds of messages per second. However depending on delays imposed by the remote entity using more than one instance MAY increase performance.

Incoming connectors must be available in more than one instance in order to allow for several simultaneous connections. When an incoming request has been processed and the remote entity logs out and drops the connection the instance can be re-used. However, in some cases if the connection fails due to a network error for example it may take a while before this is detected and the session is cleared and available for use again. In this case it would be an advantage to have more instances than the expected number of simultaneous connections. If two incoming connections are expected maybe 5 instances would be suitable.

5.6 Message types

The default message type in EMG is SMS. This for example means that when a message is received on an SMTP connector it is automatically parsed and converted to a format that suits SMS and all attachments that are not text are discarded. In order to preserve the formatting for an incoming e-mail message it is therefore necessary to set the message type to EMAIL. Then the conversion will not take place until the message is sent out over a connector of another message type.

The message types in EMG:

SMS
: Default message type

MMS
: Multimedia message (MMS)

EMAIL
: E-mail message formatted according to RFC 822 or RFC 1521 (MIME)

WAPPUSH
: WAP push message

DLR
: Delivery report

CDRUPDATE, CDRCANCEL
: CDRs (only used with EMG Roamer)

FAX
: Reserved for future use

5.7 Mappings

In order to translate between different character sets, being able to send and receive specific symbols etc it may be necessary to apply mappings to the message data.

Mappings in EMG are defined per connector using the MAPPING keyword. The keyword specifies a filename which contains the mapping or translation table. EMG handles one-to-one, one-to-many, many-to-one and many-to-many.mappings and both text and binary data can be processed.

The format of a mapping file is the following (fields are tab-separated), left-hand column specifies a source data looked for and right-hand column the replacement:

```
# This is a comment in a sample mapping file.

# First we define a mapping applied when a message is received
# Both character string and hex codes can be used
# First line translates all “a”s into “b”s
```

```
IN <
“a”        “b”
“xyz”      “zyx”
03,”a”     “a”,03,02
>

# Then a mapping which will be applied to outgoing messages
First line will have no effect (as long as ascii char 0x41 is “A”
OUT <
“a”,41,”b” “aAb”
01,02,03   20,20
>
```

Mappings are processed top to bottom and the following two examples will generate different result if applied to the string “aabacc”.

```
# Example 1 (will generate “ddbdcc”)
IN <
“a”        “d”
“aa”       “xx”
>

# Example 2 (will generate “xxbdcc”)
IN <
“aa”       “xx”
“a”        “d”
>
```

Incoming (IN) mappings are applied when an incoming message is parsed before it is logged and routed. Therefore the EMG server only sees the translated message data.

Outgoing (OUT) mappings are applied to the message just before they are sent and leave EMG. Therefore the translated message data is never seen in the EMG server.

Characters can be removed by using an empty replacement, ““.

By default mappings are applied to all messages. However, usually you would not want mappings to be applied to binary or unicode (UCS2) messages. These mappings can be suppressed by using the connector keywords NOBINARYMAPPING and NOUCSMAPPING

5.8 Address rewriting

Both source and destination addresses can be rewritten by EMG after a message has been received or before a message is sent. This is particularly useful to make sure that addresses comply with the format requirements of the receiving entity.

There are four keywords used for destination address rewriting applied in the order they appear below:

- REMOVEPREFIX
- REQUIREPREFIX
- REPLACEPREFIX
- REGEXP_DESTADDR (introduced in EMG 3)

There are also four corresponding keywords for source address rewriting:

- REMOVEPREFIX_SOURCEADDR
- REQUIREPREFIX_SOURCEADDR
- REPLACEPREFIX_SOURCEADDR
- REGEXP_SOURCEADDR (introduced in EMG 3)

These keywords handle outgoing messages via a specific connector while the corresponding keywords ending with "_IN" handle incoming messages, as in REPLACE_PREFIX_IN and REGEXP_SOURCEADDR_IN.

REPLACEPREFIX was introduced in EMG 2.5 and can do all that the REMOVE/REQUIRE-keywords can do and more. It takes as argument one or more comma-separated pattern/replacement pairs where each matched (prefix) pattern is replaced by the corresponding replacement. An empty pattern can be used to add a prefix and an empty replacement to remove a prefix. All pairs are applied sequentially in the order they appear.

An example:

REPLACEPREFIX=+/,00/,0/46

This example could be used in Sweden where we want to handle addresses received in the following formats:

+46123456	International format, + as prefix
46123456	International format, no prefix
0046123456	International format, 00 as prefix
070123456	National format

When messages are sent out the receiving SMSC expected format is usually an international format without any prefix. The above REPLACEPREFIX statement would perform the wanted rewrite. Once again, operations are performed in sequence from left to right.

First any prefix ('+' or '00') is removed if present, then if a national number was supplied the leading 0 is replaced by the country code (46). If instead a +-character was required for the connection only a small modification would be needed:

REPLACEPREFIX=+/,00/,0/46,/+

This example performs the same conversion as the earlier and in addition a "+" is added as the final step.

5.8.1 Masquerading

Address masquerading can be used to hide the real sender of a mobile originated (MO) message before forwarding the message to a third-party. This may be required by some operators in order to get permission to handle MO traffic.

The masquerading is performed using a simple but efficient obfuscation of the address and the algorithm used is reversible and unique for each address. An address with 1-9 digits will be translated into a 10-digit address. Addresses with 10-18 digits will be translated into a 20-digit address.

When using the masquerade keyword on a connector it will be applied to all messages sent and received over that connector.

See the connector keyword MASQUERADE for more information.

5.8.2 Source Address Translation (SAT)

Source Address Translation (SAT) is the procedure of replacing the source address of a message with another address and is most commonly used for implementing bidirectional e-mail to SMS services.

For an e-mail to SMS service an incoming e-mail will obviously have an e-mail address as source (sender) address and before forwarding the message to a mobile phone as SMS it is needed to replace the e-mail address with a valid GSM number. If the user should then be able to reply to the message by simply hitting the reply button the GSM number much be choosen so that the reply message can be routed back to EMG via the SMSC.

Further if the combination of source and destination address can be made unique for each message a specific mobile phone user receives it will be possible to map a specific reply to a message back to the original message sent to the user. We call this threaded messages.

All this functionality can be implemented using SAT pools in EMG where a SAT pool is a pool of source addresses from which EMG chooses an address for each message going out to a mobile phone via EMG.

For the connector where the e-mail is to be received the connector keyword SATPOOL_CREATE_IN is used to indicate that SAT entries should be created when messages are received. On the connector where a reply SMS arrives the SATPOOL_LOOKUP_IN keyword is used in order to perform the lookup for the previously created SAT entry and to rewrite the address back to the original e-mail address.

The SATPOOL_CREATE and SATPOOL_LOOKUP keywords works the same way but operates on messages being sent out via the connector where they are used.

Sample configuration (parts of configuration omitted):

```
SATPOOL sat1 <
ADDRESSRANGE=10010-10020
THREADED
>

CONNECTOR smtp-in1 <
PROTOCOL=SMTP
...
SATPOOL_CREATE_IN=sat1
# Keep source e-mail address intact
REGEXP_SOURCEADDR_IN=
ROUTE=smsc
...
>
```

```
CONNECTOR smsc <
PROTOCOL=SMPP
...
SATPOOL_LOOKUP_IN=sat1
ROUTE=smtp-out1
...
>

CONNECTOR smtp-out1 <
PROTOCOL=SMTP
INSTANCES=10
ADDRESS=#MX
DOMAIN=example.com
MAPPING=mappings/hso-iso8859-out.map
DEFAULT_MSGTYPE=EMAIL
>
```

5.9 Inheritance and virtual connectors

When having a large configuration with many connectors there will be many connectors with similar configurations. In order to minimize configuration and make it more readable it is possible to let connectors inherit attributes from other connectors and also defining virtual connectors which are only used for inheritance and never instantiated themselves.

Sample configuration:

```
# Parent for all outgoing SMSC connectors
CONNECTOR smsc-template <
TYPE=OUTGOING
INSTANCES=1
NOBINARYMAPPING
LOGMESSAGE=160
LOGPDU
VIRTUAL
>

CONNECTOR smsc1 <
INHERIT=smsc-template
ADDRESS=10.0.0.1:5000
PROTOCOL=SMPP
USER=user1
PASSWORD=secret
>

CONNECTOR smsc2 <
INHERIT=smsc-template
ADDRESS=10.0.0.2:5000
PROTOCOL=UCP
USER=user1
PASSWORD=secret
>
```

A connector can only inherit from one other connector but it is possible to have “inheritance chains” where connectors inherit in multiple levels. If a keyword is specified on both a parent connector as well as the child connector the keyword on the child connector is used.

5.10 Limiting connector queue sizes

By using the general keyword MAXTOTALQUEUESIZE it is possible to limit the total number of messages in the connector queues server-wide. This is useful when EMG is one of several components through which messages will pass and it is preferable not to let queues build in EMG. When the queue size limit has been reached EMG will reject further incoming messages using a, protocol specific, temporary error code, indicating that the client should check back later for a new try.

It is possible to use the IGNOREMAXTOTALQUEUESIZE keyword for a connector in order for it to accept messages even after the limit has been reached. This is useful for testing purposes as well as for priority connectors.

MAXTOTALQUEUESIZE does not impose an exact limit. It may be exceeded by 10-20 messages depending on current server conditions.

5.11 Sample configurations

Sample connector configurations.

5.11.1 Incoming MGP supporting up to 3 connections

```
CONNECTOR mgp-in1 <
TYPE=INCOMING
PROTOCOL=MGP
ADDRESS=localhost:7185
# Up to 10 connections
INSTANCES=3
# File where we find authentication info for users
USERS=/etc/emg/mgp-users
>
```

5.11.2 Incoming SMPP supporting up to 10 connections

```
CONNECTOR smpp-in1 <
TYPE=INCOMING
PROTOCOL=SMPP
ADDRESS=localhost:9000
# Up to 10 connections
INSTANCES=10
# File where we find authentication info for users
USERS=/etc/emg/smpp-users
>
```

5.11.3 Incoming CIMD2 supporting 1 connection

```
CONNECTOR cimd2-in1 <
TYPE=INCOMING
PROTOCOL=CIMD2
ADDRESS=localhost:9000
# One connection only
INSTANCES=1
# File where we find authentication info for users
```

```
USERS=/etc/emg/cimd2-users
# Force messages to be routed to connector ebe1
ROUTE=ebe1
>
```

5.11.4 Outgoing CIMD2

```
CONNECTOR smsc1-cimd2 <
TYPE=OUTGOING
PROTOCOL=CIMD2
# Connector to server 10.0.0.1 port 5000
ADDRESS=10.0.0.1:9000
# One instance is enough
INSTANCES=1
# Username/password to use when authenticating
USERNAME=cimd2user
PASSWORD=secret
# Try to connect one time before considered connection failed
MAXFAILEDCONNECTS=3
# When connection failed sleep for 5 minutes before trying again
MAXFAILEDSLEEP=300
# If destination address does not start with “00”, add it
REQUIREPREFIX=00
# IDLETIMEOUT defaults to 30 seconds
>
```

5.11.5 Outgoing UCP using authentication via operation 60

```
CONNECTOR smsc1-ucp <
TYPE=OUTGOING
PROTOCOL=UCP
# Connector to server 10.0.0.1 port 5000
ADDRESS=10.0.0.1:5000
# One instance is enough
INSTANCES=1
# Username/password to use when authenticating
USERNAME=ucpuser
PASSWORD=secret
# Type of number, short number alias (used by auth operation 60)
AUTHTON=6
# Number plan id, private(used by auth operation 60)
AUTHNPI=5
# Do not timeout when idle
IDLETIMEOUT=0
# Incoming messages should be routed to connector smpp-in1
ROUTE=smpp-in1
>
```

5.11.6 Outgoing UCP via modem

```
CONNECTOR smsc2-ucp <
TYPE=OUTGOING
PROTOCOL=UCP
# Phone number to operator’s modem pool
ADDRESS=01122334456
# Modem is connected to /dev/ttyS0 (Linux?)
MODEM=ttyS0
```

```
# Init string to send before dialing
INIT_STRING=AT&F&K3
# One instance is all a tty can handle
INSTANCES=1
# Wait 2 seconds after connect
WAITDELAY=20
# Wait 2 seconds after message operation is sent
MSGDELAY=20
# Operator only allows us to send 2 messages per session
OPS_MAXPERSESSION=2
# If message does not contain a source address use this
DEFAULT_SOURCEADDR=7654321
>
```

5.11.7 Outgoing HTTP

```
CONNECTOR http1 <
TYPE=OUTGOING
PROTOCOL=HTTP
# We use full URL for HTTP
ADDRESS=http://www.myhost.com/cgi-bin/handlemessage.sh
# One instance is enough
INSTANCES=1
# Timeout after 5 seconds idle
IDLETIMEOUT=5
>
```

5.11.8 Outgoing EBE

```
CONNECTOR ebe1 <
TYPE=OUTGOING
PROTOCOL=EBE
# Program or script to execute
ADDRESS=/usr/local/bin/ebe-handlesms.sh
# One instance is enough, if more we need program to be thread-
safe
INSTANCES=1
>
```

5.11.9 Outgoing GSM

```
CONNECTOR gsm1 <
# GSM connectors are always outgoing
TYPE=OUTGOING
PROTOCOL=GSM
# TTY to which GSM device is attached
MODEM=cua/a
# Only one instance is allowed
INSTANCES=1
# Poll for incoming messages every 30 seconds
POLLRECEIVE=30
# No SCA in PDU, needed for some GSM devices
#GSMNOSCA
# Memory storage is used (Ericsson devices)
#GSMSTORE=ME
>
```

5.11.10 Outgoing OIS

```
CONNECTOR ois1 <
TYPE=OUTGOING
PROTOCOL=OIS
ADDRESS=10.0.0.1:5480
# Only one instance is allowed
INSTANCES=1
>
```

6. Routing

There are three different possibilities that determines what connector a message will be routed via.

They are (highest priority first):

- Any routing criteria specified for a message
- Any routing criteria specified for a connector
- The routing table

The routing table is definitely the preferred method since it provides functionality for load-balancing, failover and can be changed for a running server without having to restart the server.

Delivery reports (DLRs) are routed a little differently. They are routed back the same way as the original message from which the DLR was requested was received unless a ROUTEDLR keyword is defined on the connector where the DLR is received or if there is a ROUTEDLR keyword specified for the user that requested the DLR.

6.1 Specifying a routing criteria for a message

This can be accomplished by using the ROUTE message option. However, it can only be used when using the MGP, HTTP and SMTP protocol to send a message, since CIMD2, SMPP and UCP do not provide mechanisms for setting routing parameters. For example, from the command-line:

```
# emgsend -o ROUTE=cimd2-1 109878 "Test message”
```

6.2 Specifying a routing criteria for a connector

This can be accomplished by using the ROUTE keyword in the server configuration file for a specific connector.

```
ROUTE=cimd2-1
```

Please note that the route only affects incoming messages via the connector. Outgoing messages are not affected by the ROUTE keyword since a routing decision has already been made prior to the message being sent via the outgoing connector.

6.2.1 The REDIRECT keyword

Some connectors can only send or received messages (uni-directional) such as SMPP 3.3 or HTTP connectors for example. Say that a DLR is routed back to a HTTP receive-only connector because the original message was received there the DLR would stay in the queue forever (or until it expires). To solve this problem it is possible to specify a REDIRECT keyword on the connector meaning that any message routed to the connector will be redirected to the connector specified by REDIRECT instead.

6.3 Routing table

The routing table is loaded from the file pointed to by the ROUTING keyword in the server configuration file.

The format of this file is 2 or 3 tab-separated fields per row: <name of incoming connector><TAB><name of outgoing connector(s)[<TAB><options>]

Comma "," is used to separate entries when there is more than one entry per field.The options field is optional and the options currently recognized are "LB" (Load Balancing), “KEYWORD” (see Keyword-based routing below), “KEYWORDSESSION”, “USERNAME”, “URL” and “PLUGINARG”.

Example:

```
smpp-in1 smsc1-ucp
smpp-in2 smsc2-smpp,smsc3-smpp
mgp1 smsc4-ucp,smsc5-ucp LB
```

The first line specifies that incoming messages on connector smpp-in1 should be routed to the connector smsc1-ucp, the second line that messages on smpp-in2 should be routed to smsc2-smpp and if it is not available or in state ERROR it should route it via smsc3-smpp (fail-over). Finally the third line is similar to the second line with the exception that if both smsc4-ucp and smsc5-ucp are available load-balancing will take place and outgoing messages will be sent by alternating between the two connectors specified.

When using failover/load-balancing a maximum of 16 outgoing connectors can be specified.

The routing table also supports routing on source or destination address prefixes. This can be specified by, in the incoming connector field (the first field) a “<“ character (source address) or “>” character (destination address) followed by a prefix. Multiple destination address prefixes can also be supplied in a file when using the “@” character. A “%” character is equivalent to a “>” character (destination address.

Example:

```
>4670123456 smsc1-ucp
@/etc/emg/smsc2-prefixes smsc2-smpp
mgp1 smsc4-ucp,smsc5-ucp LB
```

When checking the routing table EMG starts with the first entry and continues down until it finds a matching criteria. The following is an example of a bad routing table where line 2 and 3 will have no effect at all since all entries that match line 2 or 3 also matches line 1 and its position in the file gives it highest precedence.

```
%0 smsc1-ucp
%012 smsc2-smpp
%0123 smsc3-ucp
```

6.4 Failover

If more than one connector is specified in the routing table and the first connector fails for some reason then the next specified connector is used. This mechanism is called failover. Failover occurs when a connector goes into state ERROR or DEAD and more than one connector is specified. During the failover the queue for the failing connector is moved to the next working connector. If there are no connectors alive at the moment then the queue entries remain in their current connector queue and waits for a connector to come back to life.

Sample configuration in routing table:

```
smpp-in1 smpp-out1,smpp-out2
```

6.5 Load balancing

When specifying load balancing for a group of connectors the connectors will be used in a round-robin manner. That is, the first message will be sent via the first specified connector, next message via connector 2 and so on. If a connector is in state ERROR or DEAD it will not be used for the load balancing until it comes back into an active state.

Sample configuration in routing table:

```
smpp-in1 smpp-out1,smpp-out2 LB
```

6.6 User-based routing

Routing can be done based on authenticated user by using the ROUTE or ROUTING keywords in the users file. If specified it will override any other routes specified for the connector or in the general routing table. However, message-specific routes will have the highest priority.

Sample users file:

```
# User-specific routing table
user1     secret          ROUTING=/etc/emg/routing.user1
# User-specific route
user2     secret          ROUTE=cimd2-ep
```

For more information check general keyword ROUTING and connector keyword USERS in appendix.

Routing to a specific user

When multiple clients connect through the same connector in order to receive messages it is important that each user receives only the messages addressed to that specific user and not any other user messages.

It is possible to route messages to a specific user by using the USERNAME keyword in the routing table.

```
>100      ucp-in          USERNAME=user100
>101      ucp-in          USERNAME=user101
```

The above will route messages based on destination address prefixes to that messages with a destination address startint with “100” will be routed to user “user100” while destination addresses starting with “101” will be routed to user “user101”.

Please note that users defined as “ADMIN” in the users file will receive all messages regardless of what user the messages are addressed to.

6.7 Keyword-based routing

When accessing services via SMS from a mobile device (Mobile Originated or MO messages) it is sometimes useful to make routing decisions based on a keyword in the message. This can be accomplished in different ways. Either all messages can be routed to an EBE connector where a shell script or program can parse the message and take action accordingly or EMG can directly parse the message and route it to different connectors depending on the destination address and a keyword in the message. The second method is more scalable due to the overhead for EBE connectors when creating new processes.

In order to use keyword-based routing in EMG the following is required:

- Routing is done via the routing table, there must be no ROUTE keyword on the connector or in the message since they will take precedence.
- The keyword must contain letters and digits only.
- The option PARSEMESSAGE must be used in the connector configuration where the message is received in order for EMG to be able to extract the keyword correctly.

Please note that after the message has been parsed for the keyword, the keyword will still be a part of the message body and is not removed from the message.

Keywords are case insensitive.

6.7.1 Sample scenario

In a sample scenario we want to set up a service where customers can request their current account balance by sending an SMS to a specific recipient number and including a keyword, “BALANCE”.

Incoming messages, which are received from a Nokia SMSC via CIMD2, should be routed to a HTTP connector that forwards messages including the keyword to a HTTP server which processes the message and takes proper action.

Messages which contain the wrong keyword is bounced to the user with an error message.

6.7.2 Setting up the connectors

Outgoing connector where we connect to a Nokia SMSC using CIMD2 and receive messages:

```
CONNECTOR nokia-cimd2 <
TYPE=OUTGOING
PROTOCOL=CIMD2
```

```
ADDRESS=10.0.0.1:9971
INSTANCES=1
USERNAME=user
PASSWORD=secret
STATIC
KEEPALIVE=60
IDLETIMEOUT=0
# Message format is “<keyword> <rest of message>”
PARSEMESSAGE=KEYWORD MESSAGE
>
```

Outgoing connector that forwards messages to a cgi-script that integrates with the application:

```
CONNECTOR myapp-http <
TYPE=OUTGOING
PROTOCOL=HTTP
ADDRESS=10.0.0.1:8080
SUBADDRESS=http://localhost:8080/cgi-bin/myapp.cgi
INSTANCES=1
>
```

The “bouncer” connector which sends a message back to the user indicating that an invalid keyword was used:

```
CONNECTOR bouncer-ebe <
TYPE=OUTGOING
PROTOCOL=EBE
ADDRESS=/usr/local/bin/bouncer.sh
INSTANCES=1
>
```

6.7.3 Setting up the routing table

The routing table is processed top to bottom which means that we should include the keyword-related entry before any more general entries:

```
# Route message from everyone incl “BALANCE” to HTTP connector
nokia-cimd2               myapp-http      KEYWORD=*:BALANCE
# Route other messages to “bouncer”
nokia-cimd2               bouncer-ebe
```

The keyword KEYWORD takes two arguments, destination address, which can include wildcards, followed by the actual keyword. The keyword can also be a ‘*’ which represents any keyword.

6.7.4 Keyword sessions

It is possible to use what is called keyword sessions meaning that a specific keyword starts a session and subsequent messages from the same subscriber need not start with the keyword while the session is active. This is particularily useful for chat services and similar.

Sample configuration:

```
nokia-cimd2  myapp-http  KEYWORD=*:CHAT,KEYWORDSESSION=3600
```

The argument to the KEYWORDSESSION keyword is the session lifetime in seconds. So, the session lifetime in the above example would be 1 hour (3600 seconds).

6.8 Concatenated Message Routing (CMR)

Since SMS are restricted to 160 characters (7-bit text) longer messages can consists of multiple messages which are assembled in the phone to one message, a concatenated message, larger than the 160 character limit. For example this can apply to ringtones, images and WAP push messages.

Sometimes it is necessary to make sure that the different parts of a concatenated message is routed via the same connector. We refer to this as CMR, Concatenated Message Routing.

CMR is enabled per connector using the CMR connector keyword in the server configuration file. Technically CMR checks the message UDH to see if the message is part of a concatenated message.

7. Sending messages

The messaging protocols, including the proprietary MGP, all include functionality to send, receive, query and delete messages.

In general, messages are kept in the queue for the connector through which it should be sent until successfully sent or until a permanent error has occurred. Most errors are considered temporary though and will cause the message to stay in queue for further delivery attempts. When a temporary error has occurred for a message it is put in the end of the queue in order for other messages to be delivered meanwhile. It is possible to specify a maximum number of delivery attempts after which the message will be considered undeliverable and will be removed from the queue.

7.1 Using HTTP connector

When EMG sends a message to an HTTP server or similar it uses the URL defined by the SUBADDRESS keyword and adds options as HTTP parameters to form a complete GET or POST request depending on the DEFAULT_SMSCOP setting for the connector.

The response to the request must be 200 (OK) in order for the message to be considered delivered successfully.

When sending binary messages or UDH the message data will be hex encoded.

EMG supports persistent HTTP/1.1 connections.

EMG supports HTTP basic authentication using the XAUTH* keywords.

7.2 Using SMTP connector

When EMG sends a message using an outgoing SMTP connector this is done via a standard SMTP session using some special mappings for the options associated with the message.

SOURCEADDR
: Put into the MAIL FROM address.

DESTADDR
: Put into the RCPT TO address.

MESSAGE
: Put into the DATA body.

All other message options are put into the message header as X-EMG-Option-<option> user-defined headers.

There are also a few specific connector keywords used by the outgoing SMTP connector:

QUOTEDSUBJECT=<c1c2>

This indicates that the message may include a subject and that the subject is enclosed using the characters c1 and c2 respectively.

Example:
If the keyword is used as follows:
QUOTEDSUBJECT=()
and the message body is
(This is the subject) ...and this is message body
the subject of the message will be set to “This is the subject” and the message body will be “...and this is the message body”.

Outgoing messages are MIME encoded as text/plain using the ISO8859-1 character set.

EMG supports persistent SMTP connections using the RSET command.

EMG supports SMTP authentication using the XAUTH* keywords.

7.2.1 Sample outgoing SMTP session

Remote SMTP server lines begin with ‘>’.

```
>220 host.domain.com ESMTP
HELO nordicmessaging.se
>250 Please to meet you
MAIL FROM:<123456@nordicmessaging.se>
>250 Sender ok
RCPT TO:<987654@domain.com>
>250 Recipient ok
DATA
>354 Enter mail, end with “.” on a line by itself
From: 123456@nordicmessaging.se
To: 987654@domain-com
X-EMG-Option-SOURCEADDR: 12345

This is the actual message!!!
.
>250 Message accepted for delivery
QUIT
>221 Closing connection
```

All message options can be used using the “X-EMG-Option-xxx:" user-defined headers. The UDH, if present in the message, will also be sent using a user-defined header with the UDH data hex encoded.

If successful, the message id is returned in the SMTP DATA command response.

When sending binary messages or UDH the message data must be hex encoded.

7.3 Using EBE connector

The EBE connector does not really send the message but rather forwards it to a script or program which will be executed with message information on standard

input. This enables the user to invoke external programs and facilitates integration with third-party solutions.

The format of the message information will be two fields per row, where the first field is the message option key (a numeric value) and the second field is the actual value of the option. The fields are tab-separated.

```
1<TAB>1003
34<TAB>127.0.0.1
30<TAB>emguser
2<TAB>123456
8<TAB>56789
16<TAB>414243
```

The first row indicates that the message id is 1003 (message option 1). The second row that it was received from a client that connected from the IP 127.0.0.1 (localhost). See MGP options chapter for more information about message options.

The exit code of the script is used to indicate whether the delivery was successful or not. An exit code of 0 indicates success, 1 a permanent error and all other exit codes temporary errors (message is kept in queue for new retries).

Sample script:

```
#!/bin/sh
tmpfile=/tmp/ebe.$$
# Default exit code is 0 (OK)
ret=0
# Write message information to file.
# If cat fails set exit code to non-zero
cat >$tmpfile || ret=1

exit $ret
```

7.4 Using emgsend

The utility emgclient implements the MGP protocol and can be used to send messages from the command-line. It checks client.cfg and command-line arguments for server, port and authentication information. The assigned message id will be displayed on standard output by default.

Example sending a plain text message using emgsend.

```
# emgsend -username emguser -password secret -o ROUTE=smsc1-ucp
070123456 “This is a test message”
```

Example sending a ringtone using to a Nokia phone (Nokia Smart Messaging). It is addressed to port 5505 in the phone and CHARCODE=2 indicates a binary message. The message is more than 160 octets so it will be split and 2 SMS will be sent to the phone.

The command-line is split over multiple lines. There is no space between the last digit and the trailing backslash.

```
# emgsend -hex -o SOURCEPORT=0 -o DESTPORT=5505 \
-o CHARCODE=2 461234567 \
0C010605041581158102 4A3A6505899195B185E995C80400E4D9\
0413220B1106889268495624B31261892CC4956249B124889348\
493624B31269892CC493424D112610922C495624C31255892284\
9A224B210718926C4966249B107109304495420D31249892AC49\
1620D21258892AC496624AB124D89248496424AB1259892AC493\
620D2124D0924C493624AB124D8926400001
```

Example sending Over The Air settings to an Ericsson T68 phone. It is addressed to port 49999 in the phone and CHARCODE=2 indicates a binary message. The message is more than 160 octets so it will be split and 2 SMS will be sent to the phone.

```
# emgsend -hex -o SOURCEPORT=49154 -o DESTPORT=49999 \
-o CHARCODE=2 461234567 \
01062C1F2A6170706C69636174696F6E2F782D7761702D70726F\
762E62726F777365722D73657474696E67730081EA01016A0045\
C60601871249018713110331323 32E34352E362E370001871C11\
03736F6E6F666F6E2E636F6D0001872270010186071103687474\
703A2F2F7761702E646B0001C6080187151103414243000101C6\
7F018715110357617000018717110368747 4703A2F2F7761702E\
646B00010101
```

7.5 Using emgclient

The utility emgclient can be used to send, query and delete messages and a few other administrative tasks. Online help is available.

7.6 Sending MMS

In order to send an MMS message you need an incoming connector that can handle incoming MMS as well as an outgoing MMS connector to a MMS center.

It is possible to use an incoming MGP connector to have EMG receive an MMS compiled by the EMG mmscomp utility and then sent using emgsend.

Sample configuration:

```
CONNECTOR mgp-mms <
TYPE=INCOMING
ADDRESS=127.0.0.1:7185
PROTOCOL=MGP
INSTANCES=5
USERS=users
DEFAULT_MSGTYPE=MMS
ROUTE=mm7
>

CONNECTOR mm7 <
TYPE=OUTGOING
ADDRESS=http://mmsc:12000/mmshandler
INSTANCES=1
VASID=myuser
VASPID=mysecret
>
```

You can compile an MMS into binary format util mmscomp:

```
mmscomp -totype PLMN -to 467012345 -o sample.mms test.txt
```

and then send it using emgsend:

```
emgsend -file sample.mms 467012345
```

8. Receiving messages

The messaging protocols, including the proprietary MGP, all include functionality to send, receive, query and delete messages.

8.1 Using HTTP connector

Sending a message from a HTTP client to the incoming HTTP connector involves issuing a GET or POST request with the proper message options as arguments. The document part of the request must be /bin/send. Sample URL for sending an SMS with the text "Hello world":

```
http://localhost/bin/send?SOURCEADDR=123456&DESTADDR=876543&MES-
SAGE=Hello+world
```

If successful, the message id is returned in the body of the HTTP response. If multiple DESTADDR are supplied in the same request multiple message ids will be returned in the same order as the DESTADDR options were supplied.

EMG supports persistent HTTP/1.1 connections.

When sending binary messages or UDH the message data must be hex encoded.

The DESTADDR parameter can occur multiple times in one request and therefore it is possible to perform batch sending with very high throughput (more than 5.000 messages per second).

The ROUTE and ROUTEDLR options all need the ALLOWROUTE keyword to be set on the connector in order to prevent users from setting these options by default.

There is no limitation in EMG on the size of an incoming HTTP POST request.

8.2 Using SMTP connector

When receiving a message via an incoming SMTP connector EMG will (by default) parse incoming message data (RFC 822 or MIME messages), discard all present non-text attachments and convert message data to the GSM character set.

This is the default behavior if connector is missing DEFAULT_MSGTYPE or it is set to NORMAL. If DEFAULT_MSGTYPE=EMAIL is used the e-mail message will be parsed but message content will remain intact. This should be used when e-mail messages should pass transparently through EMG.

A few message options will be set from SMTP provided information:

SOURCEADDR
: Retrieved from MAIL FROM address.

DESTADDR
: Retrieved from RCPT TO address.

MESSAGE
: Retrieved from the DATA message body.

EMG will also look for user-defined headers on the format “X-EMG-Option-<option>:”. If present they will be used and override any equivalent information provided in the SMTP session.

There are also a few specific connector keywords used by the incoming SMTP connector:

USESUBJECT
: This indicates that the Subject of the message should be included short message.

USESENDER
: This indicates that the sender of the message should be included short message.

SEPARATOR
: Specifies which separator should be used between sender, subject and message body. Default: “ “ (a single space)

USEPRIORITY
: This indicates that the X-Priority message header, if present, will be used to set the priority of the short message

All message options can be used using the “X-EMG-Option-xxx:" user-defined headers. The UDH, if present in the message, will also be sent using a user-defined header with the UDH data hex encoded.

As many messages as possible will be sent in each SMTP session using the RSET command between the messages.

When sending binary messages or UDH the message data must be hex encoded.

8.2.1 Sample incoming SMTP session

Remote SMTP client lines begin with ‘>’.

```
220 <hostname> ESMTP
>HELO <client host>
250 Pleased to meet you
>MAIL FROM:<user@domain.com>
250 ok
>RCPT TO:<123456@domain.com
250 ok
>DATA
354 Enter mail “.” ends
>From: “User Smith” <user@domain.com>
>To: <123456@domain.com>
>Subject: Test SMS
>Date: Thu, 14 Feb 2002 17:36:00 +0100
>MIME-Version: 1.0
>Content-Type: text/plain
>  charset=”iso-8859-1”
>Content-Transfer-Encoding: quoted-printable
```

```
>X-Priority: 3
>
>Test
>.
250 Messaging accepted (id=7398)
>QUIT
221 Closing connection
```

8.2.2 Preventing SMTP relaying

By specifiying which IP addresses are local (and trusted) and what domains are handled locally EMG can be set up to reject unauthorized relaying.

Sample configuration:

```
CONNECTOR smtp-in1 <
TYPE=INCOMING
..
LOCALIPS=127.0.0.1/32
LOCALIPS=10.0.0.0/8
LOCALIPS=192.168.0.0/16
LOCALDOMAINS=example.com.
...
>
```

The above would indicate that only clients connector from private networks would be able to send e-mail to other domains than example.com through EMG.

If a client connecting from another IP tries to send e-mail to a domain other than example.com it would receive a “Relaying denied” error message in the SMTP session.

8.3 Receiving WAP push via PAP

EMG can act as a PPG (Push Proxy Gateway) and handle incoming WAP push messages issued via PAP (Push Application Protocol), a protocol based on XML over HTTP.

Sample configuration:

```
CONNECTOR pap <
TYPE=INCOMING
ADDRESS=127.0.0.1:8088
PROTOCOL=PAP
INSTANCES=5
ROUTE=smsc
REDIRECT=pap-dlr
>

CONNECTOR pap-dlr <
TYPE=OUTGOING
# Dummy address, real address will be in DLR
ADDRESS=http://localhost
PROTOCOL=PAP
INSTANCES=1
>
```

When a WAP push is received via PAP it will be compiled into binary form and for the outgoing SMS message the UDH parameters source port and destination port will be set. This allows for the WAP push to be delivered via SMS to the recipient transparently.

It is possible to request that a delivery confirmation should be delivered back to a specific URL. This is handled by the REDIRECT keyword and the pap-dlr connector so that any DLRs routed back to the pap connector will be redirected to the outgoing pap-dlr connector and since a URL will be specified in the PAP request that URL will be used instead of the address specified on the pap-dlr connector.

9. Delivery receipts (DLR)

9.1 Overview

Delivery receipts, or DLRs as we call them, are special status messages generated by the SMSC when the final status of a message has been reached. For example if a message is sent to a phone successfully the SMSC can generate a DLR when the phone has acknowledged receiving the message.

Usually when a message passes through EMG and it is successfully delivered to the SMSC the status of the message is RELAYED. This means that the message was successfully transferred to and accepted by the SMSC but we still do not know if the message reached its final destination, the phone.

However if a DLR was requested then the SMSC would send a DLR back to EMG when the final status is reached, in this case DELIVERED. EMG then updates the message status from RELAYED to DELIVERED, a final status.

A DLR basically consists of a message id and a status. In EMG the MGP option MGP_OPTION_MSGTYPE is set MSGTYPE_DLR (5) to indicate that a message is in fact a DLR.

DLRs are propagated through EMG as any other message except that the special ROUTEDLR keyword is used for routing. If ROUTEDLR is not specified an DLR is routed to the connector from which the original message was received.

Also for messages terminated within EMG for any reason DLRs will be generated and sent out. For example, when a message is delivered using an HTTP, SMTP, MGP or EBE connector EMG sets the final status to DELIVERED and a DLR is generated, if requested.

9.2 Requesting a DLR

A DLR can be requested in several ways.

9.2.1 Using emgsend

The MGP option DLR can be specified on the command line.

```
emgsend -o DLR=1 46123456 “test message”
```

9.2.2 Using connector keyword

If DEFAULT_DLR keyword is used the default behavior for messages using the connector is set. However, a message-specific option always supersedes a connector option.

```
CONNECTOR smpp-out <
...
DEFAULT_DLR=1
...
>
```

The keyword FORCE_DLR on the other hand overrides any DLR message option already set. When FORCE_DLR=1 is used on a connector where messages are received the DLRs received or generated will be forwarded back to the client, while if FORCE_DLR=1 is used on a connector where messages are sent the received DLRs will only be used to update the message status and then silently dropped.

Please note that DLRs silently dropped will not appear in the corresponding connector log but will only show up in the pdu log for the connector.

9.3 Routing DLRs

For incoming DLRs routing decisions are made a little differently than for normal messages.

It is evaluated in the following order with the highest priority first:

- ROUTEDLR specified for user (in users file or in DB) who sent the original message for which DLR was requested
- ROUTEDLR specified on original message
- ROUTEDLR specified on connector where DLR is received/generated
- ROUTEDLR specified by general keyword in server.cfg
- DLR is routed back to the connector where the original message was received (and optionally redirected via connector keyword REDIRECT)

9.4 DLRs and buffered status

Some SMSCs send a DLR when the message has been accepted or if the message is buffered in the SMSC due to handset being switched off for example.

Such a DLR will not be propagated through EMG but will only update the “bufferedstatus” field in the routelog table within EMG. When using EMG with a database this field can be queried in order to determine if and why a message has been buffered.

The message status will remain as “RELAYED” even after a DLR with a buffered status has been received.

10. Logging

Logging is performed by the EMG log server which runs in the emgd process but in a separate thread. It implements a special log queue in order to minimize impact on EMG performance.

The log files are located in the EMGDIR/log directory and there are different files for different logs:

general
: The general log file where server-wide information is stored. Debug information will also be in this file.

connector.*
: Connector-specific log files where connections and messages sent and received will be placed.

security
: Security related information such as invalid username/password combinations, blacklist/whitelist information etc.

pdu.*
: Connector-specific log files where all protocol operations will be logged. This is the more protocol-specific version of the connector log introduced in EMG 3.

The log level determines how detailed the logging should be and can be set both for the server-wide logs and for the connector-specific logs using the LOGLEVEL keyword in the general or connector context.

When the server is started with the -debug or -debug2 option the log level is set to DEBUG or DEBUG2 and the log messages are also displayed on stdout. This can be useful for debugging purposes.

10.1 Location of log files

Log files are by default put in the directory EMGDIR/log. The default location of this directory would be /etc/emg/log which normally will be in a root partition with limited disk space.

There are two ways to change the location of the log files:

- Create a log file directory and make a symbolic link from EMGDIR/log to the new log file directory. This must be done while the server is stopped.
- Starting with EMG 2.5 it is also possible to set an environment variable EMGLOGDIR, before starting the server, pointing to the directory where the log files should be placed.

It is also possible to use log file rotation based on size in order to limit the maximum amount of disk space occupied by log files.

10.2 Format of log files

All log files contain a timestamp with a precision down to one msec. It also contains a message text and, if relevant, message properties. The default format is similar to the syslog format and does not include year. The general keyword LOGYEAR changes the date format in the timestamp to YYYY-MM-DD.

10.2.1 Connector log file

The connector log file includes all events for a specifiec connector. Specifically all messages sent and received will show up in one of the connector log files. A message which passes through EMG will show up at least two times usually in different log files. One entry when message was received and one entry when message was sent. It is possible for the same message to occur more than twice in log files, for example if sending the message first failed and the a retry was made which was successful. It is possible to differ between final and non-final entries since only the final entries include the ENDSECS (95) and ENDMSECS (96) message options.

In EMG 2.5 the connector log file format changed slightly to be easy to parse. Format is one event per row:

```
<timestamp> (<instance>) <event> <OK/ERR> [(<info>)] [<options>]
```

timestamp
: Mandatory. Default format is MMM DD hh:mm:ss.mmm. If LOGYEAR is used format is YYYY-MM-DD hh:mm:ss.mmm.

instance
: Mandatory. Connector instance on which the event occurred.

event
: Mandatory. Type of event that has occurred. Can be of one the following values:
CONNECT, DISCONNECT, LOGIN, LOGOUT, SEND, RECEIVE, REJECT, MAXFAIL, EXPIRE

OK/ERR
: Mandatory. Indicates whether the event was successful or failed

info
: Optional. Information related to the event. A text string consisting of one or more comma-separated fields on a key[=value] format where the value part is optional and may be surrounded by quotation marks, “. Possible key values are: pdu, dlr, orphaned and info.

  Example:

  SEND ERR (pdu=1/1,info=”72”):

  This would indicate a SEND event failed. The pdu sent was one out of one and the protocol specific error code was 72.

options
: Optional. If the event is related to a message, the message MGP options would appear here on a key:value format.

10.2.2 Sample incoming connector log file

```
Sep 11 22:34:34.306 (2) CONNECT OK (info=”127.0.0.1”)
Sep 11 22:34:34.316 (2) LOGIN OK (info="emguser")
Sep 11 22:34:34.433 (2) RECEIVE OK (orphaned) 001:71 034:127.0.0.1
022:stenor 059:mgp 093:1031776474 094:393 008:1234 017:3
Sep 11 22:34:34.468 (2) LOGOUT OK
Sep 11 22:34:34.471 (2) DISCONNECT OK
```

The log file entry for an incoming MGP connector above shows that an incoming connection from localhost (127.0.0.1) was received, the user “emguser” logged in successfully and one message was received. The message could not be routed and was therefore orphaned. All message options are displayed on a key:value format, for example the recipient MSISDN, 1234, is indicated by MGP option 8 (MGP_OPTION_DESTADDR). Also the message body is not logged, only the message length, three characters, indicated by MGP option 17.

When an operation fails it will be indicate by the text ERR accompanied with a protocol-specific error code within the parenthesis.

10.2.3 Sample outgoing connector log file

```
Sep 11 22:35:28.664 (4) CONNECT OK
Sep 11 22:35:29.071 (4) LOGIN OK
Sep 11 22:35:30.275 (4) SEND OK (pdu=1/1) 001:71 034:127.0.0.1
022:stenor 059:mgp 093:1031776474 094:393 008:1234 095:1031776530
096:269 017:3
Sep 11 22:35:40.512 (4) LOGOUT OK
Sep 11 22:37:29.007 (4) DISCONNECT OK
```

The log file entry for an outgoing CIMD2 connector above shows that a connection has been made with a sessionid of 4 and then one message consisting of one PDU has been sent successfully to the end-point. A number of message parameters can also be seen on the format “key:value”.

When an operation fails it will be indicate by the text ERR accompanied with a protocol-specific error code within the parenthesis.

10.2.4 PDU log files

While the connector log files contains message related events encoded as MGP options the PDU log files contains the actual protocol-specific operations being sent and received per connector.

PDU log files will be created either if the connector log level is DEBUG or DEBUG2 or if the connector keyword LOGPDU is present for the connector in question.

10.3 Log file rotation

Log files can be rotated based on time or size using the general keyword ROTATELOGS.

10.3.1 Log file rotation based on time

When log file rotation should be accomplished based on time the log files are rotated with a specified interval. On rotation the current log files are renamed with a timestamp as suffix.

10.3.2 Log file rotation based on size

Log file rotation based on size is accomplished by renaming the log file when it has exceeded a specified size using an index as suffix. The most recently log file rotated out would have ".1" as suffix, the next most recent ".2" and so on up to the specified total number of log files (segments) to save.

10.4 Logging to a database

If a database has been prepared for use with EMG it is possible to put the connector log in the database. The connector log files will also be used but it may be convenient to add the information into a database for example for generating statistics in an efficient way.

In order to put the connector log in a database the database must be prepared and a database profile referenced using the general keyword DBPROFILE. Also the general keyword CONNECTORLOGDB needs to be added to the server.cfg file in order to indicate that the log should be put in the database.

The connector log will be stored in the table "connectorlog" which can be found in the schema file included in the EMG distribution.

The event will given using the numeric event values as indicated below:

CONNECT	0
DISCONNECT	1
LOGIN	2
LOGOUT	3
SEND	4
RECEIVE	5
REJECT	7
MAXFAIL	9
EXPIRE	10

11. Security

Security in the EMG environment is enforced in a number of ways. However, the mechanisms are available at an application level and the system as a whole including hardware, operating system etc should be secured by other means of protection.

EMG can use SSL for encryption for all protocols. However, it requires the remote entity to also support SSL for the connection in question. If SSL support is not available it is recommended to use VPN or similar for protecting data transmitted over the Internet.

For HTTP and SMTP EMG also supports the authentication mechanisms available in the protocol such as HTTP basic authentication and SMTP login/plain/cram-md5 authentication.

When creating incoming connectors make sure that the correct IP address is specified by the ADDRESS keyword. Multi-homed machines with more than one network interface may give several possibilities for the IP address specified, make sure to pick the correct one. Connectors that will only be accessed locally on the machine should use IP address 127.0.0.1 (localhost). The localhost interface can only be accessed internally on the machine.

11.1 Access control

Incoming connectors can be restricted to only accept connections from specific client IP addresses or subnets.

See ACCESS connector keyword.

11.2 Authentication

Incoming connectors usually require authentication in the form of a username and password being supplied after the connection has been established and before any messages can be sent or received. If no encryption is used this information is sent in clear text.

See USER, USERNAME, PASSWORD connector keywords.

11.3 Blacklists and whitelists

Certain source or destination addresses may need to be logged separately or even blocked (rejected). To enable this blacklists and whitelists can be used per connector or server-wide.

A blacklist can be used to specifically reject or log an address, while a whitelist, if present, will reject all addresses not in the whitelist.

If an address is rejected by any blacklist the address will be rejected since whitelists cannot be used to override the behavior from a blacklist.

Processing of blacklists and whitelists is illustrated by the pseudo code below.

```
DoLog = false
DoReject = false

If connector blacklist exists
          If entry matches
                         If action LOG -> DoLog = true
                         If action REJECT -> DoReject = true
If general blacklist exists
          If entry matches
                         If action LOG -> DoLog = true
                         If action REJECT -> DoReject = true
If connector whitelist exists
          If entry matches
                         If action LOG -> DoLog = true
          else
                         DoReject = true
If general whitelist exists
          If entry matches
                         If action LOG -> DoLog = true
          else
                         DoReject = true
If DoReject = true
          Reject message
Else If DoLog = true
          Log message and allow
Else
          Allow message
```

See BLACKLIST, WHITELIST general and connector keywords.

11.4 Using SSL

EMG supports SSL for all protocols although it is most commonly for HTTP connections. SSL provides means for authentication using certificates and encryption (with or without a certificate).

11.4.1 Outgoing SSL without a certificate

An outgoing connector, for example HTTP, can use SSL without a certificate by just adding the connector keyword SSL to the connector configuration. This will encrypt all data sent over the connection protecting it from eavesdropping. The type of encryption used will depend on what is negotiated between the client and the server.

11.4.2 Incoming SSL

Using SSL for an incoming connection requires a certificate which can be defined server-wide or per-connector. The certificate is stored in the form of a so called PEM file.

12. Database support

A number of EMG components support DB connectivity. Currently the supported databases are MySQL and PostgreSQL. However, EMG does not require a database in order to work. Some functionality in future releases may depend on the database support but the core functionality does not depend on it.

Specifically database support may affect:

- Account information
- Routing log
- Connector log
- MGP phonebooks
- EMG Roamer module

In order to use a database a few steps must be accomplished:

- A database server must be installed
- The database must be initialized for EMG
- A database profile must be created in the configuration file, server.cfg
- The database profile must be referenced in EMG

12.1 Getting started with a DB

12.1.1 Installing the database server

Please visit the website of the database supplier for more information on generic installation, configuration and licensing of the database used.

12.1.2 Initializing the database

Scripts for database initilization is provided in the EMG distribution, for example emg-schema-mysql.sql. This script does not create the database. Also it does not drop the tables if they exist. For MySQL there is a script performing these tasks included in the distribution. It is named createemgdb-mysql.sh and is provided as reference.

12.1.3 Creating a database profile

In order to use the database a database profile must be defined in server.cfg.

Sample entry:

```
DB emg <
TYPE=MYSQL
HOST=localhost
PORT=3306
DBNAME=emg
USERNAME=emg
PASSWORD=secret
INSTANCES=1
>
```

Please note that it is currently not support to set INSTANCES > 1 for a database profile.

12.1.4 Referencing a database profile

Use the DBPROFILE keyword in the configuration file, server.cfg to reference the previously created database profile (in this case called “emg”).

For EMG Roamer, the DBPROFILE keyword would be used in the roamer.cfg file.

Example:

```
DBPROFILE=emg
```

12.2 Using the database

12.2.1 Putting the message route log in a database

By adding the keyword ROUTELOGDB all necessary information about a message will be stored in the “routelog” table in the database. Together with the actual message contents which is stored in the “messagebody” table all information for tracking as well as billing messages should be readily available.

Numerous improvements of the routelog table has been made for official release of EMG 3.0.8 which implements EMG schema version 18.

12.2.2 Putting the connector log in a database

By adding the keyword CONNECTORLOGDB all connector events that are written into the connector log files are also stored in the database. For each message received by EMG and sent out there will be two corresponding entries in the connectorlog, one for each event (receive and send).

12.2.3 User authentication from database

By using the keyword USERDB instead of USERS on a connector the user information will be read from the specified database profile. User information are read from two tables “emguser” and “emguseraccess”.

Sample configuration (parts of configuration omitted):

```
DBPROFILE=emg

DB emg <
...
>

CONNECTOR smpp-in1 <
...
USERDB=emg
>
```

Minimum content in database:

```
INSERT INTO emguser (username, password) VALUES('user1','secret');

INSERT INTO emguseraccess (userid, ipaddress, ipwidth, connector)
          VALUES(1,'0.0.0.0',0,'*');
```

This would allow user "user1" to connect to any connector from any IP address. We assume that the userid assigned by the first insert is "1".

EMG will read user information from database for every login so there is no need to refresh or reload the server after adding or modifying account information.

12.3 Schema version handling

From time to time the EMG database schema is updated. This is automatically detected by the server when the server is started and if the database indicates an older schema version than is used by the server the server will refuse to start until the schema has been upgraded.

A schema upgrade is performed by executing "emgd -upgradedb". Please note that with large tables this can take a considerable amount of time, even hours with really large databases. The EMG server must NOT be running when the database schema is upgraded or the database will potentially be corrupted.

The SQL that will be executed in order to perform the upgrade can be displayed with "emgd -upgradesql".

EMG reads the current schema version from the "emgsystem" table which should holds one row with "keyname = emgschema" and "value" set to the current schema version. If the emgsystem table or the row is missing a schema version of "0" is implicated.

13. Performance

Performance is usually on top of the agenda when discussing product capabilities. We start out by defining high performance in the EMG context as the ability to process more than 100, even up to 5.000, messages per second. By processing we mean that a message is received, potentially converted and re-transmitted.

EMG is a very efficient platform using several different mechanisms in order to achieve this effectiveness and high performance:

- EMG is multi-threaded as opposed to multi-processed
  Threading is a much more efficient way to perform multiple simultaneous tasks than using traditional operating system processes. Creation, destruction and task-switching using threads is at least 100 times more efficient than the same procedures for processes.
- EMG encourages use of persistent connections
  The amount of overhead introduced by each connect, login, logout, disconnect is huge on a high-performance system.
- EMG implements efficient queue algorithms
  Often a system-wide queue or point of synchronization can be a bottleneck and turn out to severely degrade system performance. The EMG design and implementation of queue effectively addresses this.

13.1 Hardware and operating system

First of all we want to make clear that EMG can process 100+ messages per second on more or less any hardware that runs the supported operating systems. If you have an EMG server licensed for 10 mps hardware performance is not really an issue. This often means that other criteria than performance can be the determining factor for the choice of hardware. For example, in some cases less expensive low-end hardware which can be easily replaced would be preferable while in other cases a high-end server with built-in redundancy would be a better choice.

13.1.1 CPU

EMG is very CPU intensive when processing many messages per second and a 50% faster CPU will give a close to 50% better result in performance.

13.1.2 RAM

Each message takes up a certain amount of RAM. For example, 100.000 messages in queue would need approx 15 MB of RAM.

13.1.3 Disk

Disk space should be sufficient to handle log files. In order to limit disk space used by log files it is possible to use log file rotation based on size. See the ROTATELOGS general keyword.

13.1.4 Operating system

With EMG 3.0.8 and later there is little difference between different OS versions on comparable hardware.

13.2 Protocols

The different protocols supported by EMG delivers different performance. Specifically EBE and MGP are not to be used for high-performance applications. The EBE protocol forwards messages to external programs or shell scripts and for each message a process is forked in the system. EBE connectors typically delivers a maximum of 20-200 mps when forking a shell script, 40-400 mps for a Perl script and 80-800 mps for a custom binary. The proprietary protocol MGP is designed for ease of use, monitoring the EMG server etc and it is not intended to be used for high-performance applications.

All other protocols (CIMD2, SMPP, OIS, UCP/EMI, HTTP, SMTP) are capable of handling hundreds, or even thousands, of messages per second.

13.3 Instances

Each active connector can be available in one or more instances. The number of instances should be kept to a minimum and in an ideal world with one system sending messages to EMG over a persistent connection and one system receiving the message one incoming and one outgoing instance would be enough.

However, for incoming connectors you may have more than one system that needs to connect to EMG. The number of instances controls how many simultaneous connections EMG can accept. Also if a connection is terminated abnormally, network outage or similar, the instance session may not be cleared before a new connection attempt is made. Therefore the number of instances for incoming connectors should be at least the number of expected simultaneous incoming connections plus a few spare ones. If you expect one incoming connection you should allocate 5 instances.

Please note that there will be a limited number of threads available in the operating system (1024 is the default value for RedHat Linux 7). Each connector instance will use up to two threads, one for reading and one for writing. This will in practice limit the maximum number of instances that can be used in the system.

Running the command “emgd -threadcount” will count and display the maximum number of threads that can be created by a single process.

13.4 Other issues

13.4.1 Modifying message content

The only way to modify message content in EMG 1.x was to use an intermediate EBE connectors which affects maximum throughput severely. In EMG 2.5 this can be done using mappings where even a complex mapping may affect perfor-

mance only by 15-20%. From EMG 3 the EMG Plugin API can used in order to implement advanced functionality in-process using a shared library which will impose a minimal overhead.

13.4.2 Server logging and debug mode

The logging in EMG is handled by a separate thread and therefore is very efficient. However, if the queue of log events builds up quicker than it can be flushed to disk it will consume RAM, I/O and ultimately CPU resources. Specifically, when running the EMG server, emgd, in debug mode, “emgd -debug”, the impact on performance is severe.

13.5 About benchmarks

When performing benchmarks and load testing in order to verify that EMG delivers to its promise it is important to first of all make sure that the tools used are designed for the performance they are supposed to measure.

There are numerous tools freely available that are capable of measuring performance up to 10-20 mps but not more than that. For example an SMTP load tester that do not use persistent connections will not be able to measure the full capabilities of EMG.

A. Command reference

The client utilities described below can be run from the command-line prompt in the operating system. They check EMGDIR/client.cfg for information about host, port, username and password. If all or part of this information is missing from the configuration file it has to be specified on the command-line.

All command utilities display help information when executed with "-help" as argument.

```
# emgsend -help
```

The utilities will return exit codes as follows:

0	Successful
1	Unknown or unspecified error
3	Invalid login
5	Invalid argument
10	Access denied

When using Bourne shell you can display the exit code by running “echo $?” after executing a command.

Example:

```
# emgsend 12345 “Test”
758901
# echo $?
0
```

A.1 emgclient

Interactive utility that enables a user to operate on messages, display queues and more. It uses the MGP protocol to communicate with the server. To be able to perform all available tasks and display all queue and orphan entries the user needs to be defined as an administrator in the user file.

Options:

-host <host name | IP address>
    Use specified hostname when connecting to server.

-port <port number>
    Use specified port when connecting to server.

-username <username>
    Username for server authentication.

-password <password>
    Password for server authentication.

A.2 emgd

The EMG server.

When the server is stopped all connector queues are flushed to file. In EMG 1.0h and later this file is named EMGDIR/queues.dat, in earlier versions EMGDIR/flush.dat. Also the routing log is flushed to disk as EMGDIR/routelog.dat.

In EMG 3 when using the persistence option all messages, DLRs and SAT entries are persisted in the spool directory specified by general option SPOOLDIR which defaults to /var/spool/emg.

Options:

-debug
: Set log level to DEBUG and display information on stdout.

-debug2
: Set log level to DEBUG2 (even more debug info) and display information on stdout.

-fg
: Run in foreground, do not detach from tty.

-hostid
: Display hostid as emgd reads it.

-refresh
: Refresh running server.

-reload
: Refresh running server without terminating current sessions.

-stop
: Stop running server.

-threadcount
: Count the number of threads can can be created. This gives a hint on the total maximum number of instances the EMG server can handle since each connector instance requires 1 or 2 threads depending on protocol used.

-upgradedb
: Upgrade the database schema. If the number of records are large this may take several hours when altering existing tables.

-upgradesql
: Print the SQL commands that would be executed to upgrade the database schema without actually executing them.

-v
: Display version and license information.

-verify
: Parse the configuration file and report found configuration errors (without starting the server).

Examples:

```
# emgd
# emgd -debug
```

A.3 emgsend

Used to send messages using the MGP protocol.

By default the newly assigned message id is displayed when the message has been successfully sent to the server.

Options:

-host <host name | IP address>
: Use specified hostname when connecting to server.

-port <port number>
: Use specified port when connecting to server.

-username <username>
: Username for server authentication.

-password <password>
: Password for server authentication.

-b <filename>
: Read recipients from the specified file. Format of file is one recipient per line.

-file <filename>
: Read the message body from the specified file.

-hex
: Message is hex encoded.

-o
: Specify message options.

-q
: Do not display message id.

-url
: URL to use for HTTP based protocols.

Examples:

```
# emgsend 101010 "Test message"
# emgsend -hex 414243
# emgsend -o ROUTE=cimd2-1 101010 "Test message"
```

A.4 emgstat

Display information about connectors and connector states

Options:

-host <host name | IP address>
Use specified hostname when connecting to server.

-port <port number>
Use specified port when connecting to server.

-username <username>
Username for server authentication.

-password <password>
Password for server authentication.

-db
Show database instance information.

-x
Display detailed information.

Output will be in 11 columns with the following information:

NAME	Connector name
INDEX	Connector index or position server.cfg file
TYPE	IN for incoming, OUT for outgoing connector
PROTO	Protocol
INST	Number of instances
USED	Maximum number of instances that is or has been in use since system start
STATE	Connector state
QSIZE	Connector queue size
AVG 1M	Number of messages per sec processed during the last minute
AVG 5M	Number of messages per sec processed during the last 5 minutes
AVG 15M	Number of messages per sec processed during the last 15 minutes

Example:

```
# emgstat
NAME             INDEX TYPE PROTO INST USED STATE  QSIZE  AVG 1M  AVG 5M AVG 15M
mgp-in1              0 IN   MGP      1    1 BOUND      0  0.00    0.00   0.00
smpp-in1             1 IN   SMPP     7    1 BOUND      0  0.00    0.00   0.00
smtp1                2 OUT  SMTP     0    0 DEAD       0  0.00    0.00   0.00
ucp-andrew           3 OUT  UCP      1    0 IDLE       0  0.00    0.00   0.00
ucp-d2               4 OUT  UCP      1    0 IDLE       0  0.00    0.00   0.00
cimd2-euro           5 OUT  CIMD2    1    0 IDLE       0  0.01    0.01   0.00
```

A.5 mmscomp

Compile/decompile a MMS message into/from binary form.

Options:

-type <value>
Type of MMS:

Allowed values:
get
send (default)

-from <text>
Sender

-fromtype <text>
Sender type (PLMN, IPV4, IPV6, EMAIL)

-to <text>
Recipient

-totype <text>
Recipient type (PLMN, IPV4, IPV6, EMAIL)

-subject <text>
Message subject

-headers <filename>
Specified a file that contains extra message headers to be added

-protocol <text>
Output protocol (EAIF, MM1, MM7, PAP)
Default MM1

-transaction <text>
Transaction

-o <filename>
Output file

-d <filename>
Decompile the specified MMS file

-debug
Display debug information on stdout

-debug2
Display even more debug information on stdout

A.6 pushtohex

Encode a WAP push PDU according to WAP-167 and WAP-168.
The output will be hex encoded and can be used with emgsend as shown below.

Options:

-type <value>
Type of WAP push:

Allowed values:
SI - Service Indication
SL - Service Loading

-href <URL>
URL or href to use. http:// will be inserted automatically.

-hrefs <URL>
URL or href to use. https:// will be inserted automatically.

-id <si-id>
Only used for type SI.

-text <string>
Push text.
Only used for type SI.

-created <YYYYMMDD>
Creation date.
Only used for type SI.

-expires <YYYYMMDD>
Date for expiration.
Only used for type SI.

-action <value>
Allowed values (SI):
signal-low
signal-medium (default)
signal-high
signal-none
delete

Allowed values (SL):
execute-low (default)
execute-high
cache

Example (sending a WAP push to MSISDN 012345678):

```
# pushtohex -type SI -href wap.domain.com -text “Test link”
```

```
01060403AE81EA02056A0045C60C037761702E646F6D61696E2E636F6D00010
35465737420646F6D61696E000101

# emgsend -o DESTPORT=2948 -o SOURCEPORT=9200 -o CHARCODE=2 \
-hex 012345678 01060403AE81EA02056A0045C60C037761702E646F6D6\
1696E2E636F6D0001035465737420646F6D61696E000101
```

A.7 rttltohex

Encode a RTTL file as a Nokia Smart Message PDU.
The output will be hex encoded and can be used with emgsend as shown below.

Please note that the receiving phone must support Nokia Smart Messaging.

Example (sending a ringtone to MSISDN 012345678):

```
# rttltohex abdelazer.rttl

0C010605041581158102 4A3A6505899195B185E995C80400E4D90413220B110
6889268495624B31261892CC4956249B124889348493624B31269892CC49342
4D112610922C495624C312558922849A224B210718926C4966249B107109304
495420D31249892AC491620D21258892AC496624AB124D89248496424AB1259
892AC493620D2124D0924C493624AB124D8926400001

# emgsend -o DESTPORT=5505 -o SOURCEPORT=0 -o CHARCODE=2 -hex \
012345678 0C010605041581158102 4A3A6505899195B185E995C80400E4D90\
413220B1106889268495624B31261892CC4956249B124889348493624B31269\
892CC493424D112610922C495624C312558922849A224B210718926C4966249\
B107109304495420D31249892AC491620D21258892AC496624AB124D8924849\
6424AB1259892AC493620D2124D0924C493624AB124D8926400001
```

B. Configuration options

All configuration options that specifies a filename can use absolute paths or relative paths which in that case will be relative to EMGDIR (default: /etc/emg).

B.1 General options

B.1.1 BLACKLIST

Syntax: BLACKLIST=<filename>

Specifies which file contains the system-wide blacklist. The format of the file is one entry per row where each entry consists of two or optionally three tab-separated fields. The first field is the sender or source address, the second field the action and the third optional field is either SOURCEADDR or DESTADDR to specify the type of address to be blacklisted. If the type of address is omitted it defaults to SOURCEADDR.

Possible values for action are: REJECT or LOG. Lines beginning with “#” are not processed.

Example:

```
070123456<TAB>REJECT<TAB>DESTADDR
070654321<TAB>LOG
```

B.1.2 CMREXPIRE

Syntax: CMREXPIRE=<integer>

Specifies the number of seconds routing information for concatenated messages should be kept. This value should be enough for all parts of a concatenated message to pass through EMG. However, in order to minimize overhead it should be kept as low as possible.

Default value: 60

Introduced in EMG 2.4

B.1.3 CONNECTORLOGDB

Syntax: CONNECTORLOGDB

Log connectorlog to DB. Requires DBPROFILE to be set.

B.1.4 CONNECTOR_LOGLEVEL

Syntax: CONNECTOR_LOGLEVEL=<string>

Default log level for connector-related messages.

See also: LOGLEVEL, LOGPDU

Default: Same as LOGLEVEL

B.1.5 DBPROFILE

Syntax: DBPROFILE=<string>

Default database profile to use. A database profile is defined by a DB < ... > section in the server.cfg file.

B.1.6 DISABLE_CREDITS

Syntax: DISABLE_CREDITS

Disable credits handling usually enabled when using a database.

B.1.7 DLRSSIZE

Syntax: DLRSSIZE=<value>

Maximum number of DLR entries stored in the system. When this limit is reached DLR entries the oldest entries will be discarded and a DLR with DLR_EXPIRES_STATUS will be sent, default “unknown”.

Default: 100.000

Introduced in EMG 3.

B.1.8 DLRVP

Syntax: DLRVP=<value>

Specifies after how many seconds a DLR will expire if not successfully sent.

Default: 3600 (1 hour)

Introduced in EMG 2.4a.

B.1.9 DNSTHREADS

Syntax: DNSTHREADS=<integer>

Maximum number of DNS threads to use for DNS lookups.

Default value: 1

Introduced in EMG 3.

B.1.10 LOGFLUSHSIZE

Syntax: LOGFLUSHSIZE=<integer>

Specifies how many log entries should be present in log queue before they are flushed to file.

Default value: 100

Introduced in EMG 3.

B.1.11 LOGLEVEL

Syntax: LOGLEVEL=<string>

Specifies which information should be logged. When a level is specified messages related to that level or higher (more severe) is logged. INFO is most useful for ordinary production use, while DEBUG can be useful to track down problems. Errors at level WARNING and ERROR may indicate configuration problems, badly formatted incoming data etc, and usually require some action to be taken.

Values:

DEBUG2	Lowest level, volume affects performance
DEBUG	Volume affects performance
INFO	Default level
WARNING	
ERROR	Only errors or more severe information is logged
CRITICAL	

Default value: INFO

B.1.12 LOGYEAR

Syntax: LOGYEAR

When used format for date part of timestamp in log files will be according to ISO 8601, “YYYY-MM-DD hh:mm:ss.mmm”.

There are two advantages to the standard syslog format: It includes year and it is easy to sort.

Introduced in EMG 2.5.

B.1.13 MAXTOTALQUEUESIZE

Syntax: MAXTOTALQUEUESIZE=<integer>

When the total queue size for all connectors in EMG has exceeded this value EMG will deny further incoming messages temporarily until queue size decrease below the specified value.

Useful for message throttling avoiding that a large queue builds in EMG.

Introduced in EMG 2.5.

B.1.14 NOEXPIRE

Syntax: NOEXPIRE

Specifies that messages in queue and orphans should not expire in EMG based on the validity period (VP). By default if a message has a VP set and EMG cannot send it before that period of time has elapsed it will expire and be removed from the queue.

B.1.15 NOFLUSH

Syntax: NOFLUSH

When used, queues are not flushed to disk when emgd is stopped. All queue entries are lost.

In EMG 1.0h the name of the file to which entries are flushed changed name from flush.dat to queues.dat.

In EMG 3 when file persistance is used messages are stored in separate files under the directory specified by the SPOOLDIR configuration option.

Introduced in EMG 1.0h

B.1.16 NOLOGSERVER

Syntax: NOLOGSERVER

Usually logging is handled by a separate thread. However, this means that there is a small delay before the event that triggers a log message occurs until it is really logged. When tracking down some problems it is useful to have logging take place synchronously which is done using this keyword.

Introduced in EMG 2.4b

B.1.17 ORPHANSSIZE

Syntax: ORPHANSSIZE=<integer>

Maximum number of entries in orphans queue.

When the maximum size is exceeded the oldest entry is discarded.

Default value: 10000

Introduced in EMG 1.0h

B.1.18 PERSISTFILES

Syntax: PERSISTFILES

Specifies that file persistence should be used. Requires that file persistence is also present in the license key.

See also: SPOOLDIR

Introduced in EMG 3

B.1.19 PERSISTSIZE

Syntax: PERSISTSIZE=<integer>

Specifies the maximum size (in bytes) of message body that will be stored in the same persist file as the other message options. If the message size exceeds the specified value, the message body will be placed in a separate file and will only

be loaded when needed. This may save considerable amounts of memory when handling many messages with large message bodies.

Default: 1024

Introduced in EMG 3

B.1.20 ROTATELOGS

Syntax: ROTATELOGS=<integer>[:<integer>]

Rotate log files based on size or time.

When rotation should be performed based on time the specified value should be an integer followed by “s”, “m”, “h”, “d” or “w” indicating number of seconds, minutes, hours, days or weeks. The log files will be renamed by appending a timestamp to the current file name and a new output file will be created.

When rotating based on size the format is two fields colon-separated where the first fields indicates the size and the unit. The unit can be “k”, “m” or “g” for kilobytes, megabytes and gigabytes. Second field indicates the number of files to save.

ROTATELOGS=2M:10 indicates that each log file will be rotated out when the file size exceeds 2 MB and that the 10 most recent files would be saved (with suffixes 0-9).

Introduced in EMG 2.0. Size-based rotation introduced in EMG 2.5.

B.1.21 ROUTEDLR

Syntax: ROUTEDLR=<string>

Specifies to which connector DLRs should be routed. This is the default route which can be overridden by the ROUTEDLR keyword on a specific connector.

Introduced in EMG 2.4a.

B.1.22 ROUTELOGDB

Syntax: ROUTELOGDB

Specifies that route log entries should be saved in database. The route log contains status for messages.

Introduced in EMG 3

B.1.23 ROUTELOGSIZE

Syntax: ROUTELOGSIZE=<integer>

Maximum number of entries in routing log.

When using ROUTELOGDB, this value is the number of entries to keep in memory, and can be left at the default setting. Entries are never deleted from the database, regardless of this value.

When the maximum size is exceeded the oldest entry is discarded.

Default value: 10000

B.1.24 ROUTING

Syntax: ROUTING=<filename>

Specifies which file contains the routing table. The format of the file is one entry per row where each entry consists of minimum two and maximum three tab-separated fields. Lines beginning with “#” are not processed.

The fields are:

- Incoming connector or address
  If field starts with “<“ it will be matched to the source (sender) address.
  if field starts with “>” it will be matched to the destination (recipient) address.
  If field starts with “@” it is taken as a file containing destination address prefixes.
  If field does not start with any of the characters above it is taken as a name of an incoming connector.
  If the value of the field contains any of the characters “*.?[“ it is considered a regular expression.
- Outgoing connector(s)
  If more than one connector is specified fail-over and optionally load-balancing will take place for the specified connectors.
- Extra options:
  LB, KEYWORD, KEYWORDSESSION, USERNAME, URL or PLUGINARG

  LB is used to specify load-balancing between the outgoing connectors.

  KEYWORD is used for keyword-based routing and specifies a source address/keyword pair combination where source address and keyword can be “*” used as wildcard.
  Example: KEYWORD=*:BALANCE

  KEYWORDSESSION Specifies that keyword sessions should be used enabling routing of subsequent messages from the same sender to the same destination even if they do not contain a valid keyword.

  USERNAME=user Specifies that only specified user can receive the message over the connector to which the message is routed.

  URL Specifies that a specific URL should be used when message is routed to a connector that uses a HTTP-based protocol.

  PLUGINARG A string that will be set as the PLUGINARG option on the message which can be used by external plugins.

B.1.25 SERVERNAME

Syntax: SERVERNAME=<string>

Specifies server name as passed to MGP clients upon login. For example used by the EMG SNMP Agent.

Default value is EMG.

Introduced in EMG 2.5

B.1.26 SHMKEY

Syntax: SHMKEY=<integer>

Each EMG server uses a shared memory segment. In order to be able to run multiple servers on the same machine each EMG server needs a unique shared memory key.

You can run the command “ipcs -m” to see what share memory keys are in use. Values are presented in hex.

Default value is 2555674929 (decimal).

Introduced in EMG 2.3

B.1.27 SPOOLDIR

Syntax: SPOOLDIR=<directory>

Directory to use for spool files. This include queue entries, SAT entries and message id file.

Default: $EMGDIR/spool

Introduced in EMG 3

B.1.28 SSL_KEYFILE

Syntax: SSL_KEYFILE=<filename>

Specifies the server-wide PEM-file where key and certificate is stored for use by SSL connectors.

Introduced in EMG 2.0

See also: Connector option SSL_KEYFILE

B.1.29 SSL_PASSWORD

Syntax: SSL_PASSWORD=<string>

Specifies the password, if any, used for the key file.

Introduced in EMG 2.0

See also: SSL_KEYFILE

B.1.30 TABLE_PREFIX

Syntax: TABLE_PREFIX=<string>

Use specified prefix when referencing DB tables.

If prefix is defined as “emg30_” then the connector log table will be referenced as “emg30_connectorlog”. Please note that the schema SQL file needs to be edited and in order for the schema to be created accordingly.

Introduced in EMG 3.0

B.1.31 TIME_OFFSET

Syntax: TIME_OFFSET=<integer>

Specifies an offset, in minutes, to be added to the current time. Negative values are allowed. Useful, for example, when the server time is in UTC and log files etc should reflect the local time instead.

Introduced in EMG 3.0

B.1.32 WHITELIST

Syntax: WHITELIST=<filename>

Specifies which file contains the system-wide whitelist. The format of the file is one entry per row where each entry consists of two or optionally three tab-separated fields. The first field is the sender or source address, the second field the action and the third optional field is either SOURCEADDR or DESTADDR to specify the type of address to be whitelisted. If the type of address is omitted it defaults to SOURCEADDR.

Possible values for action are: OK or LOG. Lines beginning with “#” are not processed.

Example:

```
070123456<TAB>OK<TAB>DESTADDR
070654321<TAB>LOG
```

Introduced in EMG 2.0

B.2 Connector options

B.2.1 ACCESS

Syntax: ACCESS=<string>

Access control

Only valid for incoming connectors. Specifies which host(s) or subnet(s) are allowed to make connections to the connector.

Example:

```
# Only permit connections from the host 192.168.0.1
ACCESS=192.168.0.1/32
# Only permit connections from the subnets 10.2 and 10.3.1
ACCESS=10.2.0.0/16,10.3.1.0/24
```

B.2.2 ADDRESS

Syntax: ADDRESS=<string>

For protocol EBE the value is the program to be executed (including the full path to the program.

For all other protocols the value is the host and port separated with a ":".

Example:

```
ADDRESS=/opt/emg/bin/report.sh
ADDRESS=localhost:7186
```

B.2.3 ADDRESSRANGE

Syntax: ADDRESSRANGE=<integer>

Specifies the address_range parameter for SMPP bind operations.

Applies to: SMPP (Outgoing)

See also: AUTHNPI, AUTHTON

B.2.4 ALLOWROUTE

Syntax: ALLOWROUTE

When specified non-admin users will be allowed to specify a message-specific route via ROUTE keyword.

Applies to: MGP, HTTP, SMTP (Incoming)

B.2.5 AUTHCODE

Syntax: AUTHCODE=<string>

Sets the field AC (authentication code originator) in UCP submit operation.

Applies to: UCP (Outgoing, operation 51)

B.2.6 AUTHNPI

Syntax: AUTHTON=<integer>

Number-Plan Indicator (NPI) for authentication operation

Applies to: SMPP (Outgoing), UCP (Outgoing, operation 60)

See also: AUTHTON

B.2.7 AUTHTON

Syntax: AUTHTON=<integer>

Type Of Number (TON) for authentication operation

Applies to: SMPP (Outgoing), UCP (Outgoing, operation 60)

See also: AUTHNPI

B.2.8 AUTOMATICTONNPI

Syntax: AUTOMATICTONNPI

When specified EMG will try to determine correct TON and NPI for source address based on the address format.

B.2.9 AUTOMATICTONNPI_ALPHANUMERIC_NPI

Syntax: AUTOMATICTONNPI_ALPHANUMERIC_NPI=<integer>

Introduced in EMG 3.0.

B.2.10 AUTOMATICTONNPI_ALPHANUMERIC_TON

Syntax: AUTOMATICTONNPI_ALPHANUMERIC_TON=<integer>

Introduced in EMG 3.0.

B.2.11 AUTOMATICTONNPI_DEFAULT_NPI

Syntax: AUTOMATICTONNPI_DEFAULT_NPI=<integer>

Introduced in EMG 3.0.

B.2.12 AUTOMATICTONNPI_DEFAULT_TON

Syntax: AUTOMATICTONNPI_DEFAULT_TON=<integer>

Introduced in EMG 3.0.

B.2.13 AUTOMATICTONNPI_SHORTCODE_NPI

Syntax: AUTOMATICTONNPI_SHORTCODE_NPI=<integer>

Introduced in EMG 3.0.

B.2.14 AUTOMATICTONNPI_SHORTCODE_TON

Syntax: AUTOMATICTONNPI_SHORTCODE_TON=<integer>

Introduced in EMG 3.0.

B.2.15 BLACKLIST

Syntax: BLACKLIST=<filename>

Specifies which file contains the connector-specific blacklist.

Format is same as for general keyword BLACKLIST.

Introduced in EMG 2.0

B.2.16 CDMA

Syntax: CDMA

When specified contatenated message UDH will always be added to binary messages even if only one part. This is required for correct functionality when using CMDA SMSC.

B.2.17 CDRFIELDS

Syntax: CDRFIELDS=<string>

Only used by EMG Roamer.

B.2.18 CMR

Syntax: CMR

Enables Concatenated Message Routing (CMR) for connector. CMR is used for ensuring that different parts of the same contatenated message is routed via the same connector which is sometimes necessary in order for the receiving phone to be able to reassemble the message correctly.

B.2.19 DEFAULT_CHARCODE

Syntax: DEFAULT_CHARCODE=<string>

Default value for character code

B.2.20 DEFAULT_DESTADDRNPI

Syntax: DEFAULT_DESTADDRNPI=<integer>

Default value for destination address (recipient) NPI.

B.2.21 DEFAULT_DESTADDRNPI_IN

Syntax: DEFAULT_DESTADDRNPI_IN=<integer>

Same as DEFAULT_DESTADDRNPI but applies to recieved messages.

B.2.22 DEFAULT_DESTADDRTON

Syntax: DEFAULT_DESTADDRTON=<integer>

Default value for destination address (recipient) TON

B.2.23 DEFAULT_DESTADDRTON_IN

Syntax: DEFAULT_DESTADDRTON_IN=<integer>

Same as DEFAULT_DESTADDRTON but applies to recieved messages.

B.2.24 DEFAULT_DLR

Syntax: DEFAULT_DLR=<integer>

Default value for delivery receipt (DLR)

B.2.25 DEFAULT_DLRADDRESS

Syntax: DEFAULT_DLRADDRESS=<string>

Specifies a default address for delivering DLRs.

Applies to: MM7

Introduced in EMG 3

B.2.26 DEFAULT_MSGTYPE

Syntax: DEFAULT_MSGTYPE=<string>

Specifies the default message type for messages received over the connector.

Values:

NORMAL	SMS
EMAIL	Email message (RFC 822 or RFC 1521, MIME)
FAX	Reserved for future use
MMS	Reserved for future use

Default value is NORMAL.

B.2.27 DEFAULT_PROTOCOLID

Syntax: DEFAULT_PROTOCOLID=<integer>

Set default value for protocol id (GSM 3.40 TP-PID).

B.2.28 DEFAULT_QPRIORITY

Syntax: DEFAULT_QPRIORITY=<integer, 1-5>

Default queue priority for messages in EMG. The lower the value the higher priority. When queued messages are processed a message with lower priority always is transmitted before messages with higher priority. If no priority is specified a priority of 3 is assigned.

B.2.29 DEFAULT_SMSCOP

Syntax: DEFAULT_SMSCOP=<string>

Default value for SMSC operation

Protocol	Value	Operation used
SMPP	1	submit_sm
SMPP	2	data_sm (default for SMPP 3.4)
HTTP	1	GET (default)
HTTP	2	POST

B.2.30 DEFAULT_SOURCEADDR

Syntax: DEFAULT_SOURCEADDR=<string>

Default value for source address (sender)

B.2.31 DEFAULT_SOURCEADDRNPI

Syntax: DEFAULT_SOURCEADDRNPI=<integer>

Default value for source address (sender) number plan indicator (NPI).

B.2.32 DEFAULT_SOURCEADDRNPI_IN

Syntax: DEFAULT_SOURCEADDRNPI_IN=<integer>

Same as DEFAULT_SOURCEADDRNPI but applies to received messages.

B.2.33 DEFAULT_SOURCEADDRTON

Syntax: DEFAULT_SOURCEADDRTON=<integer>

Default value for source address (sender) type of number (TON).

B.2.34 DEFAULT_SOURCEADDRTON_IN

Syntax: DEFAULT_SOURCEADDRTON_IN=<integer>

Same as DEFAULT_SOURCEADDRTON but applies to received messages.

B.2.35 DEFAULT_VP

Syntax: DEFAULT_VP=<integer>

Default value for validity period (in seconds)

See also: FORCE_VP

B.2.36 DELAYFIRSTMESSAGE

Syntax: DELAYFIRSTMESSAGE=<integer>

Specifies a delay to be imposed before first message is sent over a connection.

Introduced in EMG 3.0.5.

B.2.37 DESTFULLNAME

Syntax: DESTFULLNAME=<string>

Specifies a default full name for destination address.

Applies to: SMTP (outgoing)

B.2.38 DLR_ERR_HEX

Syntax: DLR_ERR_HEX

Treat the error code “err” in SMPP delivery receipts as a hex value.

Applies both sent and received delivery reports.

B.2.39 DLREXPIRES

Syntax: DLREXPIRES=<integer>

Specifies the time, in seconds, before a DLR entry expires.

Default: 86400 (24 hours)

B.2.40 DLR_EXPIRES_STATUS

Syntax: DLR_EXPIRES_STATUS=<string>

Specifies the status to be set for messages when their corresponding DLR entry expires.

Allowed values: "UNKNOWN", "DELIVERED", "INPROCESS", "FAILED", "DELETED", "EXPIRED", "REJECTED", "CANCELED", "QUEUED", "ORPHANED", "RELAYED"

Default: “UNKNOWN”

B.2.41 DLRIGNOREKEYWORD

Syntax: DLRIGNOREKEYWORD

B.2.42 DLRMINMATCHLENGTH

Syntax: DLRMINMATCHLENGTH=<integer>

Minimum number of characters of address (starting from end) that must match when matching DLRs.

Default value: 3

Introduced in EMG 2.5b.

B.2.43 DOMAIN

Syntax: DOMAIN=<string>

If an address does not contain a domain part when being sent out over an SMTP connector the specified domain name is added.

Applies to: SMTP (outgoing)

B.2.44 FAILOVER

Syntax: FAILOVER=<string>

Specifies an alternate connector to use if an error is received for a specific message in response to the submit operation. The message is then re-routed to the specified connector.

Applies to: All protocols

Introduced in EMG 3.0.16

B.2.45 FIRST_TRN

Syntax: FIRST_TRN=<integer>

Specifies that another transaction number should be used instead of default. Applies to SMPP connectors that connect to a remote entity that requires a certain transaction number for the first transaction.

B.2.46 FORCE_CHARCODE

Syntax: FORCE_CHARCODE=<integer>

Replaces character code on the message with the specified value.

Values:

0	Default
1	IA5
2	8-bit
4	UCS2

B.2.47 FORCECLOSE

Syntax: FORCECLOSE

Specifies that the connection should be closed after the current request.

Applies to: HTTP

B.2.48 FORCE_DCS

Syntax: FORCE_DCS=<integer>

B.2.49 FORCE_DESTADDR

Syntax: FORCE_DESTADDR=<string>

If present the specified message destination address will be set to this value even if already present in the message.

B.2.50 FORCE_DESTADDR_IN

Syntax: FORCE_DESTADDR_IN=<string>

Same as FORCE_DESTADDR but applies to received messages.

B.2.51 FORCE_DESTADDRNPI

Syntax: FORCE_DESTADDRNPI=<integer>

If present the specified message destination address number plan indicator (NPI) will be set to this value even if already present in the message.

Introduced in EMG 2.4b.

B.2.52 FORCE_DESTADDRNPI_IN

Syntax: FORCE_DESTADDRNPI_IN=<string>

Same as FORCE_DESTADDRNPI but applies to received messages.

B.2.53 FORCE_DESTADDRTON

Syntax: FORCE_DESTADDRTON=<integer>

If present the specified message destination address type of number (TON) will be set to this value even if already present in the message.

Introduced in EMG 2.4b.

B.2.54 FORCE_DESTADDRTON_IN

Syntax: FORCE_DESTADDRTON_IN=<string>

Same as FORCE_DESTADDRTON but applies to received messages.

B.2.55 FORCE_DESTPORT_IN

Syntax: FORCE_DESTPORT_IN=<integer>

Set destination port (UDH) of received messages to the specified value.

Introduced in EMG 3.0.16

B.2.56 FORCE_DLR

Syntax: FORCE_DLR=<integer>

Specifies that DLRs should always be requested regardless what is set in the message received.

B.2.57 FORCE_MESSAGE

Syntax: FORCE_MESSAGE=<string>

Replaces message body with the specified message body.

B.2.58 FORCE_PRIORITY

Syntax: FORCE_PRIORITY=<integer>

Replaces message priority with the specified priority.

B.2.59 FORCE_PROTOCOLID

Syntax: FORCE_PROTOCOLID=<integer>

Replaces message protocol id (GSM 3.40 TP-PID) with the specified protocol id.

B.2.60 FORCE_SERVICETYPE

Syntax: FORCE_SERVICETYPE=<integer>

Replaces message service type with the specified service type on messages sent over the connector.

B.2.61 FORCE_SERVICETYPE_IN

Syntax: FORCE_SERVICETYPE_IN=<integer>

Same as FORCE_SERVICETYPE but applies to messages received over the connector.

B.2.62 FORCE_SOURCEADDR

Syntax: FORCE_SOURCEADDR=<string>

If present the specified message source address will be set to this value even if already present in the message.

B.2.63 FORCE_SOURCEADDR_IN

Syntax: FORCE_SOURCEADDR_IN=<string>

Same as FORCE_SOURCEADDR but applies to received messages.

B.2.64 FORCE_SOURCEADDRNPI

Syntax: FORCE_SOURCEADDRNPI=<integer>

If present the specified message source address number plan indicator (NPI) will be set to this value even if already present in the message.

Introduced in EMG 2.4b.

B.2.65 FORCE_SOURCEADDRNPI_IN

Syntax: FORCE_SOURCEADDRNPI_IN=<integer>

Same as FORCE_SOURCEADDRNPI but applies to received messages.

B.2.66 FORCE_SOURCEADDRTON

Syntax: FORCE_SOURCEADDRTON=<integer>

f present the specified message source address type of number (TON) will be set to this value even if already present in the message.

Introduced in EMG 2.4b.

B.2.67 FORCE_SOURCEADDRTON_IN

Syntax: FORCE_SOURCEADDRTON_IN=<integer>

Same as FORCE_SOURCEADDRTON but applies to received messages.

B.2.68 FORCE_SOURCEPORT_IN

Syntax: FORCE_SOURCEPORT_IN=<integer>

Set source port (UDH) of received messages to the specified value.

Introduced in EMG 3.0.16

B.2.69 FORCE_VP

Syntax: VP=<integer>

Force the validity period to be set to the specified value (seconds) for all messages that passes through the connector. To compare with the DEFAULT_VP which sets the validity period for the message only if it is missing.

Introduced in EMG 1.0n as VP, changed to FORCE_VP in EMG 2.4.

See also: DEFAULT_VP

B.2.70 GSMNOSCA

Syntax: GSMNOSCA

SCA (Service Center Address) should not be included in PDU. Needed for some GSM devices.

B.2.71 GSMSTORE

Syntax: GSMSTORE=<string>

Specifies what kind of storage should be used for messages. Ericsson devices store messages in memory and require GSMSTORE=ME.

B.2.72 HEXID

Syntax: HEXID=<integer, 0 or 1>

In SMPP 3.3 message ids are returned in submit_sm_resp operations as a hexadecimal value while in SMPP 3.4 message ids are alphanumeric. However, some SMSCs do not comply with that or are configurable. Using this behavior it is possible to toggle how EMG should treat the message ids.

Applies to: SMPP

Introduced in EMG 3.

B.2.73 HOME_IMSI

Syntax: HOME_IMSI=<string>

Only used by EMG Roamer.

B.2.74 HOME_VLR

Syntax: HOME_VLR=<string>

Only used by EMG Roamer.

B.2.75 IDLETIMEOUT

IDLETIMEOUT=<integer>

Idle timeout (in seconds). When connector has been idle for the specified period of time the connector is disconnected. If a value of 0 is specified this feature is disabled and the connector will not timeout. Please note however that the connector will detect if the remote peer is disconnected by for example a network timeout.

Default values:
10 seconds (outgoing)
0 (no timeout) (incoming)

B.2.76 IGNOREMAXTOTALQUEUESIZE

Syntax: IGNOREMAXTOTALQUEUESIZE

B.2.77 INHERIT

Syntax: INHERIT=<string>

Used to specify that connector options should be inherited from another connector (parent). A connector is only allowed to inherit from one other connector. If a connector option is present for the child connector then the> parent is ignored for that keyword.

Introduced in EMG 3.

See also: VIRTUAL

B.2.78 INITSTRING

Syntax: INITSTRING=<string>

Modem initialization string (AT-command sequence)

For dial-up connectors using a modem this keyword can be used to specify a command string to be sent to the modem before the AT dial command is sent.

B.2.79 INSTANCES

Syntax: INSTANCES=<integer, 0-999>

Number of instances for connector

A connector can exist in zero or more instances. Zero instances means the connector is dead. When a message is being sent via a connector any of the instances can be used. Normally you would specify 1 to get one instance of an outgoing connector until there is a specific reason to change this. Incoming connectors need to be available in more than one instance if you want to be able accept multiple connections simultaneously.

Please note that each connector instance will use two threads and that the total number of threads available in the operating system will be limited.

B.2.80 INTERFACEVERSION

Syntax: INTERFACEVERSION=<string>

Protocol version for the interface.

Values:
SMPP: “33” or “34” for SMPP 3.3 and SMPP 3.4 respectively.

B.2.81 KEEPALIVE

Syntax: KEEPALIVE=<integer>

Time (in seconds) between keepalive packets

Only valid for outgoing connections. Specifies how often keepalive packets should be sent to avoid connection timeout on a static connection.

See also: STATIC

B.2.82 LIBRARY

Syntax: LIBRARY=<filename>

Used for customized connector implementations.

Applies to: Prootcol DLL

Introduced in EMG 3.

B.2.83 LOCALDOMAINS

Syntax: LOCALDOMAINS=<string>

Specifies one or more domains handled by EMG and will be used together with keyword LOCALIPS to determine whether relaying is allowed or not.

Multiple occurances allowed.

Applies to: SMTP (incoming)

Introduced in EMG 3.

B.2.84 LOCALIPS

Syntax: LOCALIPS=<string>

Specifies one or more IP addresses to be considered local and will be used together with keyword LOCALDOMAINS to determine whether relaying is allowed or not.

Multiple occurances allowed.

Applies to: SMTP (incoming)

Introduced in EMG 3.

B.2.85 LOGLEVEL

Syntax: LOGLEVEL=<string>

Log level for connector-specific entries in general log file.

Values:DEBUG2, DEBUG, INFO, WARNING, ERROR, CRITICAL

See also: LOGPDU

B.2.86 LOGMESSAGE

Syntax: LOGMESSAGE=<integer>

Specifies that the message data, or part of the message data, should be logged to the connector log file. By default only the length of message is logged to the connector log file, not the contents.

The message data is hex encoded in the log file.

B.2.87 LOGPDU

Syntax: LOGPDU

At log level DEBUG or DEBUG2, the protocol data units that are sent and received by the connectors are logged to files with the names pdu.<connector-name>, in the log directory. Set the LOGPDU option to enable this log at other log levels.

Introduced in EMG 3.

B.2.88 LONGMESSAGE

Syntax: LONGMESSAGE=<integer, 0-255>

Number of messages a long message (>MESSAGELENGTH, default160, septets) can be split into.

Default value: 4

B.2.89 LONGMODE

Syntax: LONGMODE=<string>

How to handle messages that exceed the maximum message length.

The maximum length of a message is the value of MESSAGELENGTH x the value of LONGMESSAGE. If a message exceeds that length it can either be truncated or rejected.

Values: REJECT, TRUNCATE

Default value: TRUNCATE

B.2.90 MAPPING

Syntax: MAPPING=<filename>

Specifies a file containing a mapping.

B.2.91 MASQUERADE

Syntax: MASQUERADE=<string>

Introduced in EMG 3.

B.2.92 MAXFAILEDCONNECTS

Syntax: MAXFAILEDCONNECTS=<integer>

Number of connection attempts before state ERROR

See also: RETRYTIME, MAXFAILEDSLEEP

B.2.93 MAXFAILEDSLEEP

Syntax: MAXFAILEDSLEEP=<integer>

Time (in seconds) to sleep in state ERROR.

Default: 60

See also: RETRYTIME, MAXFAILEDCONNECTS

B.2.94 MAXMESSAGELENGTH

Syntax: MAXMESSAGELENGTH=<integer>

Specifies the maximum length of a message that will be accepted. Messages longer will be rejected with the SMTP error code 552.

Default value: 30000 bytes

Applies to: SMTP (incoming)

B.2.95 MAXTRIES

Syntax: MAXTRIES=<integer>

Specifies the maximum number of times delivery may fail before message is considered undeliverable.

Default value: 10

Introduced in EMG 2.5.

B.2.96 MESSAGELENGTH

Syntax: MESSAGELENGTH=<integer>

Specifies the length of a message.

Default value: 160

Applies to: CIMD2, SMPP, UCP/EMI

See also: LONGMESSAGE, LONGMODE

B.2.97 MESSAGEMODE

Syntax: MESSAGEMODE=<string>

Introduced in EMG 3.

B.2.98 MESSAGES_PER_REQUEST

Syntax: MESSAGES_PRE_REQUEST=<integer>

Introduced in EMG 3.

B.2.99 MIMEBOUNDARY

Syntax: MIMEBOUNDARY=<string>

Introduced in EMG 3.

B.2.100 MMS_TEXT_CHARSET

Syntax: MMS_TEXT_CHARSET=<string>

Introduced in EMG 3.

B.2.101 MODE

Syntax: MODE=<string>

Specifies whether a connector should be allowed to transmit (TX), receive (RX) or both transmit and receive (TRX) messages.

A connector that receives a message but is only permitted to transmit messages will reject the operation.

If a message is put in the connector queue for a connector that is only allowed to receive message the message will simply remain in the queue.

For SMPP 3.3 connectors this keyword determines whether a bind_transmitter or bind_receiver will be issued during login. For TX and TRX bind_transmitter will be used, for RX bind_receiver.

Values: RX, TX, TRX

Default value: TRX

Introduced in EMG 1.0l.

B.2.102 MODEM

Syntax: MODEM=<string>

Modem device

Used for outgoing, dial-up connectors. Specifies the tty (without the leading ‘/dev/’) to use for dialing when making a connection.

B.2.103 MSGDELAY

Syntax: MSGDELAY=<integer>

Delay between messages (in seconds)

Used to specify that a delay should be applied after a message has been sent. This can be useful or even required for low-performance links, modem connections etc.

B.2.104 MSGRETRYTIME

Syntax: MSGRETRYTIME=<integer>

Introduced in EMG 3.

B.2.105 MTBILLING

Syntax: MTBILLING=<string>

Introduced in EMG 3.

B.2.106 NOBINARYMAPPING

Syntax: NOBINARYMAPPING

Suppresses messages being applied to binary messages.

Introduced in EMG 2.5.

B.2.107 NOUCS2MAPPING

Syntax: NOUCS2MAPPING

Suppresses messages being applied to UCS2 (Unicode 16-bit) messages.

Introduced in EMG 2.5.

B.2.108 NOUSERMESSAGEREFERENCE

Syntax: NOUSERMESSAGEREFERENCE

Introduced in EMG 3.

B.2.109 OPSENTEXPIRES

Syntax: OPSENTEXPIRES=<integer>

B.2.110 OPS_MAXINCOMING

Syntax: OPS_MAXINCOMING=<integer>

B.2.111 OPS_MAXOUTGOING

Syntax: OPS_MAXOUTGOING=<integer>

B.2.112 OPS_MAXOUTSTANDING

Syntax: OPS_MAXOUTSTANDING=<integer>

Maximum number of operations outstanding.

Some of the protocols, like SMPP can handle that the remote peer sends multiple operations without waiting for a response in between. This improves performance but requires that the application keeps track of operations sent but not acknowledged (outstanding).

The actual number of outstanding operations permitted depends on the type of protocol in use, the EMG setting and also the setting in the remote application (usually the SMSC).

Default values: 1 for all protocols

B.2.113 OPS_MAXPERSESSION

Syntax: OPS_MAXPERSESSION=<integer>

Maximum number of operations to send per session

When this limit has been reached the connector will disconnect.

B.2.114 ORIGIN

Syntax: ORIGIN=<string>

Introduced in EMG 3.

B.2.115 PARSEMESSAGE

Syntax: PARSEMESSAGE=<string>

Used to parse a message for message options. The format for the supplied string is <message option>[<separator><message option>[...]]. The message is always expected to be the last option and indicates that the rest of the message contains the message body.

Example:

```
PARSEMESSAGE=DESTADDR MESSAGE
```

When a message is received it will be expected to contain the destination address followed by a space, ‘ ‘, and then the message body. In this case the destination address in the message will override any destination address supplied in other ways. The special keyword KEYWORD can be used for keyword-based routing and will be used together with keywords optionally specified in the routing table.

Applies to: All protocols (incoming)

B.2.116 PASSWORD

Syntax: PASSWORD=<string>

Used for outgoing connector authentication via the connector protocol.

Applies to: All protocols (outgoing)

B.2.117 PLUGIN

Syntax: PLUGIN=<string>

Introduced in EMG 3.

B.2.118 POLLRECEIVE

Syntax: POLLRECEIVE=<integer>

Time (in seconds) between message polls.

Indicates that the connector should connect periodically to allow for messages to be received over the incoming connector. This is used when EMG needs to connect to the SMSC in order to receive messages and the connection cannot be static.

Applies to: CIMD2 and SMPP (outgoing)

B.2.119 PREFIX

Syntax: PREFIX=<string>

Introduced in EMG 3.

B.2.120 PRESERVESAR

Syntax: PRESERVESAR

Introduced in EMG 3.

B.2.121 PROMPT

Syntax: PROMPT=<string>

Specifies a string to be sent to the client side after connect before the protocol server starts. A newline will be added after the string.

Applies to: All protocols (incoming)

B.2.122 PROTOCOL

Syntax: PROTOCOL=<string>

Protocol used by the connector.

Values: CIMD2, EBE, HTTP, MGP, MMS, OIS, SMPP, SMTP, UCP

B.2.123 QUOTEDREPLY_SEPARATOR

Syntax: QUOTEDREPLY_SEPARATOR=<string>

Specifies separator to be used between reply and original message when using quoted reply option.

Default: “--- Original message below ---”

Introduced in EMG 3.0.

B.2.124 QUOTEDSUBJECT

Syntax: QUOTEDSUBJECT=<string>

Indicates that when a message is going to be sent out using SMTP the message subject is part of the message. The supplied string must consist of 2 characters where the first character indicates

Example:

```
QUOTEDSUBJECT=()
```

Message: “(This is the subject) And here comes the message body”

Will extract the string within parenthesis from the message above and use it as the subject of the message.

Applies to: SMTP (outgoing)

Introduced in EMG 2.0

B.2.125 REDIRECT

Syntax: REDIRECT=<connector>

Introduced in EMG 3.

B.2.126 REGEXP_DESTADDR

Syntax: REGEXP_DESTADDR=<regexp>

Introduced in EMG 3.

B.2.127 REGEXP_DESTADDR_IN

Syntax: REGEXP_DESTADDR_IN=<regexp>

Introduced in EMG 3.

B.2.128 REGEXP_KEYWORD

Syntax: REGEXP_KEYWORD=<regexp>

Introduced in EMG 3.

B.2.129 REGEXP_MESSAGE

Syntax: REGEXP_MESSAGE=<regexp>

Introduced in EMG 3.

B.2.130 REGEXP_SOURCEADDR

Syntax: REGEXP_SOURCEADDR=<regexp>

Introduced in EMG 3.

B.2.131 REGEXP_SOURCEADDR_IN

Syntax: REGEXP_SOURCEADDR_IN=<regexp>

Introduced in EMG 3.

B.2.132 REJECT_EMPTY

Syntax: REJECT_EMPTY

Reject messages that contains an empty message body.

Introduced in EMG 3.0.3.

B.2.133 REMOVEPREFIX

Syntax: REMOVEPREFIX=<string>

Removes prefix for destination address (DESTADDR).

The specified prefix will be removed if it exists in the destination address. It can be combined with REQUIREPREFIX and is applied before REQUIREPREFIX.

For example if REMOVEPREFIX=+ and REQUIREPREFIX=00 and the destination address is +4612345678 the '+' will be removed and the destination address will be padded with "00", the final destination address being 004612345678.

B.2.134 REMOVEPREFIX_SOURCEADDR

Syntax: REMOVEPREFIX_SOURCEADDR=<string>

Same as REMOVEPREFIX but applies to source addresses.

Introduced in EMG 2.5.

B.2.135 REPLACEPREFIX

Syntax: REPLACEPREFIX=<string>

Specifies a sequence of source/target address rewrite patterns.

B.2.136 REPLACEPREFIX_IN

Syntax: REPLACEPREFIX_IN=<string>

Introduced in EMG 3.

B.2.137 REPLACEPREFIX_SOURCEADDR

Syntax: REPLACEPREFIX_SOURCEADDR=<string>

Same as REPLACEPREFIX but applies to source addresses.

Introduced in EMG 2.5.

B.2.138 REPLACEPREFIX_SOURCEADDR_IN

Syntax: REPLACEPREFIX_SOURCEADDR_IN=<string>

Introduced in EMG 3.

B.2.139 REQUIREPREFIX

Syntax: REQUIREPREFIX=<string>

Required prefix for destination address (DESTADDR).

If specified this prefix will be added to destination addresses that do not have the prefix.

For example if REQUIREPREFIX=00 and the destination address is 4612345678 the destination address will be padded with “00”, the final destination address being 004612345678.

B.2.140 REQUIREPREFIX_SOURCEADDR

Syntax: REQUIREPREFIX_SOURCEADDR=<string>

Same as REQUIREPREFIX but applies to source addresses.

Introduced in EMG 2.5.

B.2.141 RETRYSCHEME

Syntax: RETRYSCHEME=<string>

Introduced in EMG 3.

B.2.142 RETRYTIME

Syntax: RETRYTIME=<integer>

Specifies the time (in seconds) to wait before trying a reconnect after a connect failure.

Default: 30

Applies to: All protocols (outgoing)

See also: MAXFAILEDCONNECTS, MAXFAILEDSLEEP

B.2.143 REVDLR

Syntax: REVDLR

Reverse order of source address and destination address for DLRs sent out over the connector.

Introduced in EMG 3.

B.2.144 REVDLR_IN

Syntax: REVDLR_IN

Reverse order of source address and destination address for DLRs received over the connector. This may sometime be necessary for DLR matching to work when remote entity sends addresses in reversed order.

Introduced in EMG 3.

B.2.145 ROUTE

Syntax: ROUTE=<string, connector name>

Primary connector to route messages to. This option only applies to incoming messages via that connector.

B.2.146 ROUTEDLR

Syntax: ROUTEDLR=<string, connector name>

Specifies to which connector DLRs should be routed.

B.2.147 ROUTING

Syntax: ROUTING=<string>

Introduced in EMG 3.

B.2.148 SATPOOL_CREATE

Syntax: SATPOOL_CREATE=<string>

Create a SAT entry for messages being sent over the connector.

Introduced in EMG 3.

B.2.149 SATPOOL_CREATE_IN

Syntax: SATPOOL_CREATE_IN=<string>

Same as SATPOOL_CREATE but applies to messages received over the connector.

Introduced in EMG 3.

B.2.150 SATPOOL_LOOKUP

Syntax: SATPOOL_LOOKUP=<string>

Specifies one or more SAT pools to be used for SAT lookups for messages being sent over the connector.

Multiple occurances allowed.

Introduced in EMG 3.

B.2.151 SATPOOL_LOOKUP_IN

Syntax: SATPOOL_LOOKUP_IN=<string>

Same as SATPOOL_LOOKUP but applies to messages received over the connector.

Multiple occurances allowed.

Introduced in EMG 3.

B.2.152 SCAADDR

Syntax: SCAADDR=<msisdn>

Set Service Center (SMSC) address in message PDU.

Applies to: GSM

Introduced in EMG 3.0.3.

B.2.153 SCAADDRNPI

Syntax: SCAADDRNPI=<integer>

Set Service Center (SMSC) address NPI in message PDU.

Applies to: GSM

Introduced in EMG 3.0.3.

B.2.154 SCAADDRTON

Syntax: SCAADDRTON=<integer>

Set Service Center (SMSC) address TON in message PDU.

Applies to: GSM

Introduced in EMG 3.0.3.

B.2.155 SENDERADDRESS

Syntax: SENDERADDRESS=<string>

Introduced in EMG 3.

B.2.156 SEPARATOR

Syntax: SEPARATOR=<string>

Introduced in EMG 3.

B.2.157 SERVICETYPE

Syntax: SERVICETYPE=<string>

Defines the servicetype for SMPP submit_sm and data_sm operations.

Applies to: SMPP

B.2.158 SIMULATE

Syntax: SIMULATE

Simulate connector operation. No operations are actually sent.

B.2.159 SMPPTZ

Syntax: SMPPTZ=<string>

If specified overrides any specified timezone for SMPP time fields.

Applies to: SMPP (outgoing)

Introduced in EMG 3.

B.2.160 SOURCEADDR_GSM

Syntax: SOURCEADDR_GSM

Indicates that alphanumeric source addresses should be GSM encoded before used in SMPP.

Applies to: SMPP

Introduced in EMG 3.

B.2.161 SOURCEFULLNAME

Syntax: SOURCEFULLNAME=<string>

Specifies a “full name” to be used in “From:” address field for SMTP.

Applies to: SMTP

Introduced in EMG 3.

B.2.162 SSL

Syntax: SSL

Specifies that Secure Socket Layer (SSL) should be used for the connector.

Introduced in EMG 2.0

B.2.163 SSL_KEYFILE

Syntax: SSL_KEYFILE

Specifies the connector-specific PEM-file where key and certificate is stored for use by SSL connectors.

B.2.164 SSL_PASSWORD

Syntax: SSL_PASSWORD

B.2.165 STATIC

Syntax: STATIC

Only valid for outgoing connectors. Specifies that the connector should connect on startup and stay connected. This will usually require that keepalive packets is sent periodically to avoid connection timeout. Also the IDLETIMEOUT should be set to 0 to avoid periodic disconnect + reconnects.

See also: IDLETIMEOUT, KEEPALIVE

B.2.166 SUBADDRESS

Syntax: SUBADDRESS=<string>

Defines the URL to use for the GET or POST operation. This keyword is no longer needed since the ADDRESS parameter understands and parses full URLs for addresses.

Example:

```
ADDRESS=localhost:8080
SUBADDRESS=http://localhost/cgi-bin/report.sh
```

Applies to: HTTP (outgoing)

B.2.167 SUBJECT

Syntax: SUBJECT=<string>

Sets the default subject.

Applies to: SMTP (outgoing)

B.2.168 SUPPRESS_EMGHEADERS

Syntax: SUPPRESS_EMGHEADERS

EMG will add “Received” headers to incoming SMTP messages unless this keyword is specified.

Applies to: SMTP (incoming)

Introduced in EMG 3

B.2.169 SYSTEMTYPE

Syntax: SYSTEMTYPE=<string>

Identifies the SMPP system_type field for a connector. May be involved in the authentication process for the connector.

Applies to: SMPP

B.2.170 TCPSOURCEIP

Syntax: TCPSOURCEPORT=<IP address>

For outgoing connectors the source IP address is set to the specified address. May be needed on a host with multiple addresses on network interfaces.

Applies to: All protocols

Introduced in EMG 3

B.2.171 TCPSOURCEPORT

Syntax: TCPSOURCEPORT=<integer, 0-65535>

For outgoing connectors the source port is set to the specified port. May be needed when for example an SMSC requires a specific source port for authentication.

Applies to: All protocols

B.2.172 THROUGHPUT

Syntax: THROUGHPUT=<integer, 1-1000>

Limits throughput for the connector in question. The value specified is the number of messages per second. A value of 0 means 0.5 messages per second.

Applies to: All protocols

Introduced in EMG 2.0

B.2.173 TYPE

Syntax: TYPE=<string, INCOMING | OUTGOING>

This indicates only whether the connector should accept incoming connections or initiate outgoing connections. It does not tell anything about the direction of the message flow. A message can be received on a outgoing connector and vice versa. This depends on the protocol used.

B.2.174 UDHVIAOPTIONAL

Syntax: UDHVIAOPTIONAL

Usually UDH parameters for source port, destination port and concatenated messages is encoded into the UDH and is sent as part of the message data with the UDH indicator (UDHI) set.

However, when this option is used on a SMPP 3.4 connector these UDH parameters will be sent as optional parameters instead. Some applications that implement SMPP 3.4 may require this.

Applies to: SMPP 3.4 (outgoing)

B.2.175 USEDELTIME

Syntax: USEDELTIME

When specified message delivery times (scheduled messages) will be enforced within the EMG server rather than passed through to the SMSC. That is if a message has a delivery time in the future th message will be kept in the EMG queue until the time for delivery is reached rather than the message being passed to the SMSC with a scheduled delivery time set.

B.2.176 USEPRIORITY

Syntax: USEPRIORITY

Indicates that the X-Priority message header should be considered for setting the priority of the message.

Applies to: SMTP (incoming)

Introduced in EMG 2.0

B.2.177 USERDB

Syntax: USERDB=<string>

Specifies that user information for authentication incoming connections should be retrieved using the specified database profile.

B.2.178 USERDB_FORCE

Syntax: USERDB_FORCE=<string>

B.2.179 USERNAME

Syntax: USERNAME=<string>

Used for outgoing connector authentication via the connector protocol.

Applies to: All protocols (outgoing)

B.2.180 USERS

Syntax: USERS=<filename>

The users file is used for authentication of incoming connections.

If filename starts with a slash '/' it is considered to be absolute, otherwise it is relative to EMGDIR.

The format of the file is one username/password combination per row with the fields tab-separated plus an extra optional field used as follows:

For MGP connectors:

ADMIN
: User is administrator.

CLIENTCONFIG=<string>
: String will be sent to client and can be used to affect client configuration from the server side. The client does not need to respect this field.

For all connectors:

FORCE_SOURCEADDR_IN=<string>
Force the source address for all messages to the specified address.

MAXSESSIONS=<integer>
Limit maximum number of session for the specific user.

ROUTE=<string, connector name>
Specify a connector that is the “default route” for the user. Used for user-based routing.

ROUTEDLR=<string, connector name>
Specify a connector to which to route DLRs requested by the user.

ROUTESAT=<string, connector name>
Specify a connector to which to route messages after a successful SAT lookup has been performeda.

ROUTING=<filename>
Specify a file containing a user-specific routing table. Used for user-based routing.

SATPOOL_CREATE=<string, SAT pool name>
Use specified SAT pool for messages received.

THROUGHPUT=<integer>
Limit throughput for the specific user.

Applies to: All protocols (incoming)

B.2.181 USESENDER

Syntax: USESENDER

Indicates that the message sender should be used as part of the message. The sender will be inserted before the message subject and body.

Applies to: SMTP (incoming)

Introduced in EMG 3

B.2.182 USESUBJECT

Syntax: USESUBJECT

Indicates that the message subject should be used as part of the message. The subject will be inserted before the message body but after the sender (if used).

Applies to: SMTP (incoming)

Introduced in EMG 2.0

B.2.183 VASID

Syntax: VASID=<string>

Used for MM7 SenderIdentification.

Applies to: MM7 (outgoing)

Introduced in EMG 3

B.2.184 VASPID

Syntax: VASPID=<string>

Used for MM7 SenderIdentification.

Applies to: MM7 (outgoing)

Introduced in EMG 3

B.2.185 VIRTUAL

Syntax: VIRTUAL

Specifies that the connector should not be instantiated in EMG but only used as a template for inheritance.

See also: INHERIT

Introduced in EMG 3

B.2.186 WAITBEFORECONNECT

Syntax: WAITBEFORECONNECT=<integer>

Introduced in EMG 3

B.2.187 WAITDELAY

Syntax: WAITDELAY=<integer>

Delay after connect (in seconds).

Applies to: All protocols (outgoing)

B.2.188 WAITFOR

Syntax: WAITFOR=<string>

Wait for specified prompt after connect

Applies to: All protocols (outgoing)

B.2.189 WHITELIST

Syntax: WHITELIST=<filename>

Specifies which file contains the connector-specific whitelist.

Format is same as for general keyword WHITELIST.

Introduced in EMG 2.0

B.2.190 XAUTH

Syntax: XAUTH

Specified that the built in protocol mechanisms should be used for authentication.

Applies to: HTTP (basic auth) and SMTP (plain/login/cram-md5)

Introduced in EMG 2.5.

B.2.191 XAUTHPASSWORD

Syntax: XAUTHPASSWORD=<string>

Specifies a password for protocol authentication.

Applies to: Outgoing HTTP (basic auth) and SMTP (plain/login/cram-md5)

Introduced in EMG 2.5.

B.2.192 XAUTHUSERNAME

Syntax: XAUTHUSERNAME=<string>

Specifies a username for protocol authentication.

Applies to: Outgoing HTTP (basic auth) and SMTP (plain/login/cram-md5)

Introduced in EMG 2.5.

B.2.193 XPASSWORD

Syntax: XPASSWORD=<string>

External password

Used for connector authentication when authentication is done in more than one step.

B.2.194 XUSERNAME

Syntax: XUSERNAME=<string>

External username

Used for connector authentication when authentication is done in more than one step.

B.3 DB options

Options used when specifying database profiles.

Example:

```
DB mysql-db1 <
HOST=localhost
PORT=3306
DBNAME=emgdb
USERNAME=emguser
PASSWORD=secret
>
```

B.3.1 DBNAME

Syntax: DBNAME=<string>

Database name

B.3.2 HOST

Syntax: HOST=<string>

Hostname used when connector to database.

B.3.3 INSTANCES

Syntax: INSTANCES=<integer>

Specifies how many instances should be created. Each instance represents one static connection to the database server.

Currently only the values 0 and 1 are allowed.

B.3.4 PASSWORD

Syntax: PASSWORD=<string>

Password used for authentication when connecting to the database.

B.3.5 PORT

Syntax: PORT=<integer>

Port used when connecting to database.

B.3.6 TYPE

Syntax: TYPE=<string>

Type of database.

Values:

MYSQL	MySQL
PGSQL	PostgreSQL

B.3.7 USERNAME

Syntax: USERNAME=<string>

Username used for authentication when connecting to the database

B.4 SAT pool options

Options used when defining SAT pools.

Example:

```
SATPOOL sat1 <
ADDRESSRANGE=4670001-4670010
THREADED
>
```

B.4.1 ADDRESSRANGE

Syntax: ADDRESSRANGE=<string>

The addresses to use for the pool. You can specify a comma separated list of individual numbers, a range of numbers with identical prefixes and lengths, or any combination of these. When a range is specified, the lower and the upper limit must have the exact same number of digits.

Example: ADDRESSRANGE=460001-460010,460015

B.4.2 EXPIRE

Syntax: EXPIRE=<integer>

The numer of minutes a source address will be reserved for a specific SAT entry. After the specified time it can be reused.

Default: 4320 (3 days)

B.4.3 QUOTEDREPLY

Syntax: QUOTEDREPLY=<integer>

Specifies that original message should be quoted when a reply is received via SAT. Values can be 0 (off) or 1 (on).

Default: 0 (off)

B.4.4 THREADED

Syntax: THREADED

TBD

B.5 Domain options

Options used for domain-specific behavior. Can be used to minimize risk of being classified as a spammer when delivering large amount of e-mails to recipients in specific domains.

Only applies to SMTP #MX connectors.

Example:

```
DOMAIN yahoo.com <
MAILSPERMINUTE=10
SESSION=1
>
```

B.5.1 MAILSPERMINUTE

Syntax: MAILSPERMINUTE=<integer>

Maximum number of e-mail to send per minute for domain.

Default: 0 (unlimited)

B.5.2 MAILSPERSESSION

Syntax: MAILSPERSESSION=<integer>

Maximum number of e-mails to send per SMTP session.

Default: 0 (unlimited)

B.5.3 PORT

Syntax: PORT=<integer>

Port to use when connecting to SMTP server for domain.

Default: 25

B.5.4 RETRYTIME

Syntax: RETRYTIME=<integer>

Time (in seconds) to wait if the DNS lookup does not find any valid MX hosts or if it is not possible to connect to the hosts found.

Default: 300 seconds

B.5.5 SESSIONS

Syntax: SESSIONS=<integer>

Maximum number of simultaneous SMTP sessions for domain.

Default: 1

B.6 Plugin options

Options used when defining plugins.

Example:

```
PLUGIN billing <
LIBRARY=/etc/emg/plugins/billing.so
INSTANCES=2
CONFIG=/etc/emg/plugins/billing.cfg
>
```

B.6.1 CONFIG

Syntax: CONFIG=<string>

The value to be used for the second argument to create_config(). Usually it is the name of the configuration file to be used by the plugin.

B.6.2 INSTANCES

Syntax: INSTANCES=<integer>

The number of threads to be used. If the plugin functions are not thread-safe set this to 1 which will make all calls serialized.

B.6.3 LIBRARY

Syntax: LIBRARY=<string>

The name of the shared library that implements the plugin.

B.6.4 OFFSET

Syntax: OFFSET=<integer>

Offset for numeric result and error codes.

For result codes the specified value will be added and for error codes 2 x the value will be added.

C. MGP options

The following options can be used with messages. The option name is given followed by its numeric value. The numeric value is used in log files etc.

Some options only applies to certain protocols. These options are simply ignored by protocols which do not support them.

All numeric key values that are not used are reserved for future use.

User-Data Header (UDH) options can be supplied in two ways. Some can be supplied by setting the corresponding MGP options, DESTPORT for example. It is also possible to include the UDH in the actual message data and set the User-Data Header Indicator (UDHI).

C.1 Option keys in numeric order

Key value	Option
1	ID
2	SOURCEADDR
3	SOURCEADDRTON
4	SOURCEADDRNPI
6	SOURCEPORT
8	DESTADDR
9	DESTADDRTON
10	DESTADDRNPI
12	DESTPORT
14	UDH
15	MSGTYPE
16	MESSAGE
17	MESSAGELEN
18	VP
19	DLR
20	DELTIME
21	SCTS
27	MSGCLASS
28	CHARCODE
30	USER
31	REPLYPATH
32	PRIORITY
34	REMOTEIP
38	ROUTE
43	PROTOCOLID
59	CONNECTOR
60	OUTCONNECTOR
61	STATUS
64	SMSCID
73	CONCATSMSREF
74	CONCATSMSSEQ
75	CONCATSMSMAX
79	BILLINGID
81	DLRID
93	STARTSECS
94	STARTMSECS
95	ENDSECS
96	EMDMSECS

97	NOTE
106	UDHI
107	REPLACEPID
108	LRADDR
109	LRPID
110	HPLMNADDR
111	SUBJECT
112	OTOA
113	DCS
118	QPRIORITY
119	XUSERNAME

C.2 Options reference

C.2.1 BILLINGID (79)

Billing ID

Only used with UCP 4.0

C.2.2 CHARCODE (28)

Character code of message

Coded into the DCS (Data Coding Scheme).

Values:

0	Default (GSM)
1	IA5
2	8-bit (binary)
3	ISO 8859-1 (Latin-1)
4	UCS2

C.2.3 CONCATSMSMAX (75)

GSM concatenated messages, number of messages

Encoded into UDH.

C.2.4 CONCATSMSREF (73)

GSM concatenated messages, reference number

Encoded into UDH.

C.2.5 CONCATSMSSEQ (74)

GSM concatenated messages, sequence number

Encoded into UDH.

C.2.6 CONNECTOR (59)

Connector on which message was received

C.2.7 DCS (113)

Data Coding Scheme

Defined in GSM 3.38.

C.2.8 DELTIME (20)

Delivery time

C.2.9 DESTADDR (8)

Destination or recipient address

C.2.10 DESTADDRNPI (10)

Destination address, number plan indicator (GSM 3.40)

Values:

0	Unknown (default)
1	ISDN
3	Data numbering plan (X.121)
4	Telex
5	Private
6	ERMES
15	Reserved for extension

C.2.11 DESTADDRTON (9)

Destination or recipient address, type of number (GSM 3.40)

Values:

0	Unknown (default)
1	International number
2	National number
3	Network specific number
4	Subscriber number
5	Alphanumeric
6	Abbreviated number
7	Reserved for extension

C.2.12 DESTPORT (12)

Destination port

Encoded into the UDH. Used to send binary messages like ringtones, WAP push etc.

C.2.13 DLR (19)

Delivery Receipt

C.2.14 DLRID (81)

Message id of original message

C.2.15 ENDMSECS (96)

Timestamp when message finished by EMG (millisecs)

This option only holds the milliseconds part of the time

Values: 0-999

C.2.16 ENDSECS (95)

Timestamp when message finished by EMG (seconds)

The milliseconds of this value is stored in ENDMSECS.

The value of the option is the time in seconds since 1 January 1970 (UTC).

C.2.17 HPLMNADDR (110)

Applies to: UCP

Introduced in EMG 1.0h.

C.2.18 ID (1)

Message id

This is the unique message id assigned by EMG.

C.2.19 LRADDR (108)

Last Resort Address

Introduced in EMG1.0h.

C.2.20 LRPID (109)

Last Resort Protocol Identifier

Introduced in EMG1.0h.

C.2.21 MESSAGE (16)

Message data

C.2.22 MESSAGELEN (17)

Length of message (in characters or bytes)

C.2.23 MSGCLASS (27)

Message class as defined by (GSM 3.40)

Values:

0	Flash SMS (message displayed immediately)
1	Store message (memory or SIM) (default)
2	Store message to SIM
3	TE (Terminal Equipment) specific

C.2.24 MSGTYPE (15)

Message type

Specifies the type of message.

Values:

SMS	1
DLR	5
Email	6
MMS	7

Default value: 1

C.2.25 NOTE (97)

Note

A user-specified text attached to the message. Can, for example, be used to hold extra information on the message.

C.2.26 OTOA (112)

Syntax: OTOA=<string>

Applies to: UCP

Introduced in EMG 1.0h

C.2.27 OUTCONNECTOR (60)

Name of connector on which message was sent out

C.2.28 PRIORITY (32)

Priority of message

There are differences in how priority options are implemented in the different SMSCs. However, in GSM 3.40 there are only priority or non-priority messages.

Values:

0	No priority (default)
1	Priority

Protocol-specific mappings

CIMD2	MGP
1-4	1
5-9,none	0

MGP	CIMD2
0,none	No value
1	1

SMPP	MGP
0,none	0
1-3	1

MGP	SMPP
0,none	none
1	2

UCP	MGP
0,1,none	0
2-3	1

MGP	UCP
0	none
1	2

C.2.29 PROTOCOLID (43)

Protocol identifier (GSM 3.40)

C.2.30 QPRIORITY (118)

Specifies the queue priority in EMG. A message with a lower value is always processed before a message with a higher value when message queues are processed in EMG.

Values:

1	Highest priority
2	
3	Normal
4	
5	Lowest priority

Default value: 3

C.2.31 REMOTEIP (34)

IP of the client that sent the message

C.2.32 REPLACEPID (107)

Replace PID (or RPID)

Applies to: UCP

Introduced in EMG 1.0h

C.2.33 REPLYPATH (31)

Reply path

C.2.34 ROUTE (38)

Message-specific route

The option value must be a valid connector name. The specified connector will be used instead of any routes specified in the routing table or on the incoming connector.

C.2.35 SCTS (21)

Service Center TimeStamp

C.2.36 SMSCID (64)

Message id assigned by the remote SMSC

The exact format and value of this option is protocol and SMSC dependent.

C.2.37 SOURCEADDR (2)

Source or sender address

C.2.38 SOURCEADDRNPI (4)

Source address number plan indicator

For possible values, see DESTADDRNPI

C.2.39 SOURCEADDRTON (3)

Source address type of number

For possible values, see DESTADDRTON

C.2.40 SOURCEPORT (6)

Source port

Encoded into the UDH.

C.2.41 STARTMSECS (94)

Timestamp when message received by EMG (millis)

This option only holds the milliseconds part of the time

Values: 0-999

C.2.42 STARTSECS (93)

Timestamp when message received by EMG (seconds)

The milliseconds of this value is stored in STARTMSECS.

The value of the option is the time in seconds since 1 January 1970 (UTC).

C.2.43 STATUS (61)

Message status

Values:

1	Delivered
2	In process
3	Failed
4	Deleted
5	Expired
6	Rejected
7	Canceled
8	Queued
9	Orphaned
10	Relayed
11	Unknown

C.2.44 SUBJECT (111)

Default subject.

Applies to: SMTP

Introduced in EMG 2.0.

C.2.45 UDH (14)

User Data Header

The User Data Header

C.2.46 UDHI (106)

User Data Header Indicator

Indicates that the message includes a UDH in the message data.

Introduced in EMG 1.0f.

C.2.47 USER (30)

Username for client that sent the message

C.2.48 VP (18)

Validity Period

C.2.49 XUSERNAME (119)

MGP username

Used to route messages not only to a specific connector but to a specific user.

Applies to: MGP

D. Error codes

In the connector log files protocol specific error codes may be displayed within parenthesis when send or receive operations fail. The most common of these error codes are specified below, per protocol.

D.1 CIMD2

Error code	Description
0	No error
1	Unexpected operation
2	Syntax error
3	Unsupported parameter error
9	Requested operation failed
100	Invalid login
102	Too many users with this login id
103	Login refused
300	Incorrect destination address
302	Syntax error in user data parameter
303	Incorrect user data parameter combination

D.2 SMPP

Error code	Description
0	No error
3	Invalid command id
4	Invalid bind status for given command
5	ESME already in bound state
8	System error
10	Invalid source address
11	Invalid destination address
12	Message ID is invalid
13	Bind failed
14	Invalid password
15	Invalid system id
20	Message queue full
21	Invalid system type
88	Throttling error (ESME has exceeded allowed message limits)
97	Invalid scheduled delivery time
98	Invalid message validity period

D.3 UCP/EMI

Error code	Description
1	Checksum error
2	Syntax error
3	Operation not supported by system
4	Operation not allowed
5	Call barring active
6	AdC (destination address) invalid
7	Authentication failure
22	Time period not valid
23	Message type not supported by system
24	Message too long

D.4 OIS

Error code	Description
0	Success
1	Invalid data
2	atabase (message data store) full
3	SME busy
5	Duplicate message
6	Destination unavailable
20	Call barred by user
21	Transmission error
24	Unknown subscriber
25	Call barred by network operator (destination)
26	Call barred by network operator (originator)
120	Network failure

D.5 HTTP

Error code	Description
200	Success
400	Bad request
401	Unauthorized
403	Forbidden
404	Not found
500	Internal error
501	Not implemented
301	Moved
304	Not modified

D.6 SMTP

Error code	Description
220	Service ready
250	Requested mail action ok
251	User not local will forward to
421	Service not available
450	Mailbox unavailable
451	Requested action aborted
452	Requested action not taken
500	Syntax error, command unrecognized
501	Syntax error in parameter or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
550	Requested action not taken

E. SMSC inter-connectivity checklist

When interacting with an operator in order to set up a connection to an SMSC the following checklist may be useful.

E.1 Your requirements

E.1.1 Send messages

The most common and basic functionality. Message price? Price differences whether the recipient is a subscriber of an other operator, domestic or foreign etc.?

E.1.2 Receive messages

If you want to receive messages addressing the message to the application (EMG) and being able to receive messages sent from phones with SIM cards from different operators can be problematic issues. Pricing?

E.1.3 Type of messages

Will you send plain text messages (max 160 characters) or will you need to send binary message with User-Data Header (ringtones, logos etc). WAP push messages over SMS is another type of message that requires UDH and 8-bit messages. Possibility to send more complex messages also depends on the protocol being used.

When sending text messages the character set is also relevant. What kind of national characters will be sent and how are they supported and handled?

E.1.4 Performance or message volume

What kind of message volume are you calculating with? This is usually measured in messages per second or messages per day. 1 message per second equals to 86400 messages in 24 hours. Pricing usually depends on volume to a high degree.

E.1.5 Support and service

Your operator may have different service and support plans for different types of customers.

E.2 SMSC connection

E.2.1 Type of connection

Does your operator provide TCP/IP, dial-up or X.25 connections. If relevant, how are incoming messages handled. Who initiates the connection, the ESME or the SMSC or both?
Are static connections allowed or should ESME disconnect immediately after sending message(s)? If yes, should keepalives be sent and if so, how often?

Is an idle timeout allowed, that is EMG waits for a specified period of time when the connection is idle before disconnecting?

E.2.2 Protocol

What messaging protocol is used for communicating with the SMSC. CIMD2, OIS, SMPP and UCP are the common protocols used. Protocol version is also of interest. There is for example a big difference between SMPP 3.3 and SMPP 3.4. Make sure the type of messages you want to send/receive can be transmitted using the protocol in question.

E.2.3 Performance

Are there any restrictions as how many messages per second can be sent? Can more than one connection be used? (Defined by the INSTANCES keyword. More than one instance for an outgoing connection is usually not necessary.) What is the maximum throughput the SMSC can handle?

The protocol being used may be asynchronous meaning that it may be possible to send an operation without waiting for the response to the previous operation sent. This enhances performance but makes the application a bit more complex since it must keep track of outstanding operations. However, EMG supports this and it makes sense to find out if the SMSC handles outstanding operations and if so how many.

E.2.4 Security

None of the messaging protocol mentioned above provide any means of security apart from basic authentication with plain text usernames and password. Does the operator provide other kinds of security mechanisms? In case of an TCP/IP-connection access to the service should be restricted based on the ESME source IP address. You will probably have to pay the bill if a malicious person gets hold of the username and password and sends unauthorized messages using your account.

E.3 Getting started

E.3.1 Account information

After installing EMG you need to set up an outgoing connector to the SMSC in question. In order to do this you need your account information. For a TCP/IP connection this includes:

- SMSC IP address
- SMS port
- Protocol used
- Username
- Password

Using this information you should be able to configure your connector and start the EMG server.

E.3.2 Sending the first message

When both EMG and the SMSC are set up it is time to test the connection. The best way to do this is simply to send a message from EMG routed via the outgoing connector to the SMSC. This will verify the connection to the SMSC, the authentication process and finally the delivery of a message.

The behavior in the following situations should also be verified (if relevant):

- Sending a message to an invalid destination address (recipient)
- Using national characters
- Sending long messages (more than 160 characters)
- Sending messages to foreign recipients (destination address format with country code etc needs to be determined)

E.3.3 Receiving a message

If relevant, messages should be sent from a phone to the application.

F. Change history

Changes between EMG releases.

There are three different types of releases:

Major release
: A full distribution including new functionality. Migrating from one major version to another will require a new license key.

Minor release
: A full distribution primarily including changes and bug fixes. Usually does not require a new license key.

Patch release
: A binaries-only release including changes and bug fixes. Applied to an existing installation from a full distribution. Does not require a new license key.

Changes marked with an asterisk, '*', includes a fix for a problem which potentially could cause the server to stop unexpectedly.

F.1 EMG 3.0.16.14262

Minor release.

- EMG is now available in version without SSL/encryption support
- Database **schema change** for "emguser" table (build 13555)
- Import performance for handling of incoming delivery reports
- New connector keyword FAILOVER for re-routing of message when an error is returned in response to submit operation
- New connector keywords FORCE_SOURCEPORT_IN and FORCE_DESTPORT_IN
- Fix for potential crash when orphaned messages expired
- PDU log format change for unknown fields, previously logged as "UNKNOWN:<value>", now logged as "?<key>:<value>"
- HTTP: Two potential buffer overflows eliminated *
- OIS: Multiple fixes for sending and receiving message incl UDH support
- UCP: Protocol id is now treated as string and not as numeric value
- SMPP: DLR_ERR_HEX can now be used for outgoing DLRs
- SMPP: Handle delivery receipts from Huawei SMSC correctly
- WAP PUSH SI/SL in UTF-8 is now handled correctly
- Validation of values for INTERFACEVERSION added when SMPP is used
- Signature change (2 new arguments) for DB functions "db_sendrequest" and "db_readresponse". **Plugins using these functions must be re-compiled to use the new signatures.**
- Plus som additional potential minor memory leaks and null pointers fixed.

F.2 EMG 3.0.15.12050

Minor release.

- Added support for regular expressions in routlist files
- SMPP: New connector keyword DLR_ERR_HEX which indicates “err” field in incoming delivery receipts should be treated as hex
- SMPP: Handle delivery receipts from Comverse SMSC correctly
- UCP: Empty messages bodies was not handled *
- Problem with refresh/reload with connectors using SSL fixed

F.3 EMG 3.0.14.11726

Minor release.

- New general keyword DISABLE_CREDITS, which disables credit handling
- SMPP: Possible data corruption when parsing invalid pdus fixed *
- SMPP: Preserve DCS value when GSM MWI is used
- PAP: Improvements for PAP proxy possibilities
- UCP: New MGP option XSER_EXTRA for handling custom XSER fields
- DLR: Possible memory corruption under high load eliminated *
- DLR: Logic rewrite for performance improvement
- New MySQL driver for Solaris 10 x86
- Check for expired DLRs only once per minute
- Ensure connection is closed if we cannot write

F.4 EMG 3.0.13.10913

Minor release.

- New connector keyword OPS_MAXEXPIRED, which controls how many outstanding operations are allowed to expire before connection is closed. Default is 0.
- PAP: Missing “Content-type” could cause server to crash *
- SMPP: Make delivery report parsing a bit more flexible
- MGP: Fail message if UDH is not correctly encoded
- Enable forwarding of DLRs for temporary status by setting bit 2 for DLR option (DLR=2 for temp only or DLR=3 for temp and permanent status)
- Routing files was not checked properly by “emgd -verify”
- Fix in retry scheme handling which would cause action for one message to be applied to multiple messages
- “Holding” a connector which did not exist would cause server to crash *
- Minor memory leak when using PLUGINARG option
- DB: Schema change (19) in routelog, “npdus” now defaults to “1”
- DB: Schema change (20) in routelog, new field “origid” which is set to the id of the original message for DLRs
- DB: Schema change(21), new table “monthlysummary” for aggregated

F.5 EMG 3.0.12.10307

Minor release.

- PAP: Fix for respecting SI/SL
- SMSCI: Do not use mod 256 on transaction number

- Concatenated messages: Never convert from UCS2 to IA5 even if one part will fit, since this will corrupt message
- Concatenated messages: Ensure we do not split a message in the middle of a multi-byte character
- AUTOWAPPUSH_IN now supports a text in addition to the URL. The first URL found in the message will be extracted and the rest of the message used as the text
- Do not update routelog table with NULL values
- Bodies for large messages was not removed when using message persistence

F.6 EMG 3.0.11.9982

Minor release.

- CIMD2: TAB is not allowed in usernames
- HTTP: HTTP status 406 was incorrectly sent for messages with empty message body
- MGP: QPRIORITY can now be set by ADMIN users
- SMPP: Added support for parsing of Comverse DLR format
- SMPP: Field data_coding was not always set correctly
- SMPP: Fix for handling of messages longer than 255 bytes
- SMTP: E-mails with leading space in message body was not handled correctly
- SMTP: Functionality for “compressing” a Unicode message into GSM IA5, if it fits, improved
- SMTP: QUOTEDSUBJECT was not handled correctly
- UCP: “/” is not allowed in usernames
- UCP: Ensure correct length is used for billingid (max 20 chars)
- smtpload: More than 10 threads was not handled
- UDH calculation could sometimes be incorrect for long messages if “concat” attributes already present in UDH
- New protocol implemented, SMSCI (CIMD2 variant)
- Loglevel now reset to “info” during reload if not present
- Increased buffer size for network communication for improved SMTP performance
- Memory leak when clearing messages and PERSISTFILES in use
- Directory structure is now only traversed every minute when using PERSISTFILES

F.7 EMG 3.0.10.9505

Minor release.

- HTTP: Validate hex encoded message bodies
- MM7: Namespace added to DeliverRsp
- SMPP: Possible null pointer when receiving invalid data *
- emgclient: Invalid pointer problem when running “deleteall” on server with a lot of activity *
- PARSEMESSAGE did not handle all IA5 characters properly

- New keyword AUTOWAPPUSH_IN for receiving a link via SMTP and converting it into a WAP Push (SI).
- Improved performance of hex encoder/decoder
- Incoming connectors should not be set to “hold” on message errors

F.8 EMG 3.0.9.9284

Minor release.

- MGP: Return code from plugin call before_receive was not respected
- SMPP: enquire_link is not allowed if not logged in
- Optimizations in DLR handling
- Optimizations in logging
- Minor memory leak when database operation aborted unexcpectedly

F.9 EMG 3.0.8.9115

Minor release.

- SMPP: Messaging mode part of parameter esm_class was not respected
- SMPP: Timezone was not handled correctly for validity period
- SMPP: Handle DLRs with BUFFERD or ACCEPTD status
- SMPP: Optional parameters with a length > 128 were not handled properly*
- UCP: Message body was hex encoded twice when logged
- HTTP: Read to “end of file” if content-length is missing
- HTTP/MGP: Handle incoming DLRs
- HTTP/MGP: Now it is allowed to set XUSERNAME option if ALLOW-ROUTE is specified
- HTTP/EBE: New connector keyword USERESPONSE which allows for a message id to be read
- SMTP: Potential null pointer when message body empty which would cause crash on Solaris*
- EBE: Improved handling of childs in order to avoid zombie processes
- MM7: Reordered some fields
- MM7: Fixed memory leak when converting from MIME to MM7
- MM7: new protocol variant MM7-TMOBILE for interoperability with T-Mobile MMC
- emgstat: Display more of connectorname if screen is wider than 80 characters
- mmscomp: Fix potential crash
- Outgoing messages were counted twice for some protocols
- Keyword CONNECTORLOGDB can now be used for connectors as well as in general section
- New connector keyword LOGDB_EVENTS for specifying which connector events should be written to connectorlog table in database
- New connector keyword NODLR to suppress DLRs being generated from HTTP, SMTP, EBE connectors
- New connector keyword CHARCODE_CONVERT which enables automatic conversion between IA5 and Latin-1 for messages sent over connector

- New DB keyword SOCKET for connecting to MySQL databases using a UNIX socket instead of TCP/IP. As a consequence PORT now must always be specified
- Automcatic conversion of UCS2 messages which does not use the “high byte” to Latin-1 added
- DLRs received with a buffered status are now handled and will update the “bufferedstatus” field in routelog accordingly
- Default lifetime for a DLR entry increased to 73 hours
- Default value for IDLETIMEOUT for outgoing connectors is now KEEPALIVE + 10 seconds
- GSM modems would not work correctly if emgd had more than 64 file descriptors open*
- New column “msgtype” in routelog table. Now routelog can be used for all message statistics without using connectorlog
- DB: Multiple schema changes for routelog
- DB: Warn if INSTANCES > 1 for a DB profile since this is not supported
- Plugin API: Added functions plugin_thread_start and plugin_thread_stop for thread-specific handling when using plugins
- Plugin API: Plugins using more than 1 instance were not thread-safe*
- Performance: Optimization of allocated buffers has increased performance substantially for this release
- Performance: Switch to mtmalloc routines on Solaris which is much faster for multi-threaded applications
- Solaris 10 x86 is now supported

F.10 EMG 3.0.7.8470

Minor release.

- Updated OpenSSL version to 0.9.7i
- Unnecessary counting of DLRs could cause high load on DLR intensive systems
- Environment variable EMGLOGDIR did not work properly
- Use of MAXSESSIONS in users file could cause deadlock *
- Eliminated potential buffer overflow in UDH parsing *
- Users connecting and then disconnecting without logging in could cause thread to be left hanging
- DCS value for IA5 message was sometimes set to 1 instead of 0
- SMTP: FORCE_CHARCODE=UCS2 would cause server to crash *
- UCP: MT_ALERT_RESP was incorrectly decoded
- pushtohex: WAP opaque data was not correctly encoded

F.11 EMG 3.0.6.8403

Minor release.

- MIME: Handle UTF-8 in MIME trees properly
- CIMD2: Protocolid was not respected
- SMPP: Potential buffer overflow when parsing pdu *
- SMPP: Send error “invalid command length” on invalid pdu length

- SMPP: Reject unsupported operations
- SMPP: Handle generic_nack properly
- SMPP: Added support for vendor-specific optional parameters
- UCP: Potential buffer overflow when parsing pdu *
- UCP: Protocolid was not respected
- MM1: Potential null pointer problem fixed *
- MM1: Use sender timestamp for date field
- MM1: Force read-reply and delivery-report to false if message-class=auto
- MM7: Handle “env:Client” as a generic client error
- MM7: Duplicate “<Priority>” tags was sometimes sent
- MM7: Use transid “0” in reply if transid is missing in request
- MM7: 5xx responses was not handled correctly
- emgclient: delete operation could delete wrong message
- Option MAXTOTALQUEUESIZE was not respected for most protocols
- Added MGP option DESTNETWORK which can be used for MNP routing
- URLs that starts with “https” in (SUB)ADDRESS should give SSL
- Algorithm for splitting long messages did not always work correctly
- Now IDLETIMEOUT is used when waiting for HTTP responses
- Minor memory leak fixed related to using invalid connector keywords
- Warn if KEEPALIVE is set to a greater value than IDLETIMEOUT
- Possible deadlock when authentication users from file and using maxsessions
- Session counter did not always handle disconnecting users correctly
- Support for multiple RETRYSCHEME keywords per connector
- Warn if DEFAULT_MSGTYPE is supplied with an invalid message type
- Potential invalid reads on big-endian archs (Solaris) *
- Plugins: Implementation of ZMRDB (MNP in Germany) and T-Mobile (Germany) billing plugins

F.12 EMG 3.0.5.7719

Minor release.

- Treat HTTP status 202 (accepted) as delivery ok
- SMTP: Error 450 is message specific and should not block entire domain for retry
- SMTP: There was a 30 second hard-coded “idle timeout” for incoming connections
- Prune /dlr and /sat directories when using persistence
- Syntax \xx can be used for binary data in keywords
- MSGTYPE=DLR was not sent for DLRs over SMTP
- EBE connector did not count used instances correctly
- New connector keyword DELAYFIRSTMESSAGE which causes a delay before the first message is sent over a connection
- Relative SMPP time did not respect days, months or years when specified
- Added concept of “weak” keywords which has a lower priority than keyword sessions via WEAKKEYWORD in routing file
- User-specific force of source address did not always work as expected
- Avoid null pointer when emgstat is used immediately after server startup *
- Binary messages should not be parsed for keywords

- Delivery time was not handled correctly for CIMD2, SMPP and UCP
- Protocol ID (GSM 3.40 TP-PID) was not handled correctly
- Keyword sessions could generate high load due to inefficient tree traversal
- Keyword sessions was only checked for messages containing a keyword
- SMPP message length was incorrectly set on big-endian platforms
- DLR: Expire field was not persisted and therefore lost on server restart
- UCP: Number of sessions was not always decreased on disconnect
- UCP: Message type was not set correctly for UCS2 and 8 bit messages via UCP deliver operation
- HTTP: Fix UDH parse problem for incoming requests
- HTTP: Avoid possible null pointer *
- MM7: Send UTF-8 data without conversion using encoding quoted-printable
- MMS: New connector option MMS_TEXT_CHARSET used to set character set for text parts in MMS
- DCS parameter was not handled correctly for HTTP
- MM1: Set status in replies
- Log plugin rejects in connector log
- Minor perfomance optimization in hex encoder
- Build date is now included in version information
- Arcor fixes
- DB schema change to version 15

F.13 EMG 3.0.4.6983

Minor release.

- New connector keyword ROUTING for specifying a connector-specific routing table
- New connector keywords REVDLR and REVDLR_IN for switching source and destination address for sent and received DLRs respectively
- New connector keywords used for setting TON and NPI when detecting address types automatically: AUTOMATICTONNPI_SHORTCODE_TON, AUTOMATICTONNPI_SHORTCODE_NPI, AUTOMATICTONNPI_ALPHANUMERIC_TON, AUTOMATICTONNPI_ALPHANUMERIC_NPI, AUTOMATICTONNPI_DEFAULT_TON, AUTOMATICTONNPI_DEFAULT_NPI
- New connector keyword REGEXP_KEYWORD for applying regular expressions to keywords
- Added new MGP options for SMPP fields source_subaddress, dest_subaddress and its_session_info
- Apply mappings to keywords as well as message body
- MMS: Keyword parsing for MMS messages implemented, first check subject and if subject does not exist use first word in first plain text message part
- MIME: Ensure quoted-printable lines are CRLF terminated
- MM7: Put short codes in ShortCode tag correctly
- MM7: Allow empty Subject tag
- Handle empty SMTP messages
- Fixed potential crash for invalid formatted HTTP cookies *
- Fixed potential crash when DNS lookup failed *

- Fixed potential buffer overflow in MM1 parser *

F.14 EMG 3.0.3.6691

Minor release.

- New connector option REJECT_EMPTY for rejecting messages with empty message body
- CMR: Fix for CMR where messages could be routed under high server load. Ported from EMG 2.5t
- DLR: Problem with UCP DLR matching when id overflow
- DLR: Fixed problem with DLR matching when FORCE_DESTADDR was used
- GSM: New connector options for protocol GSM: SCAADDR, SCAADDRTON and SCAADDRNPI used to set SMSC address in message PDU
- GSM: Always use capital letters when hex encoding in driver for GSM modems
- HTTP: Always URL encode MGP option values sent over HTTP
- MGP: Requesting route log via MGP could cause emgd to crash *
- MIME: Do not insert CRLF before boundary attribute in Content-Type header
- MIME: Handle when space is missing after ":" in headers
- MIME: Potential buffer overflow in MIME part ids *
- MMS: Force connection close when using HTTP 1.0
- MM7: Various MM7 fixes
- SAT: QUOTEDREPLY did not work properly
- SAT: Subject was not saved properly when using SAT with SMTP
- ARCOR: Added Arcor multimessage packet logging
- Orphans: A small ORPHANSSIZE was used emgd could crash on server startup *
- Authentication: When authenticating from database emguser and useraccess was joined improperly allowing access to server from any IP address
- Authentication: Separate warning message when user has exceeded max number of simultaneous sessions
- Credits: Improved handling of credits charging 1 credit when receiving message and later charge additional credits if message was split into multiple PDUs
- Refresh: Connector hold flag was reset during server refresh/reload
- License: On Linux when MAC address of eth0 is not available use IP address for license check instead
- Added reasontext for CIMD2, SMPP and UCP for protocol specific error information
- Fixed minor memory leak when incoming connections was terminated prematurely

F.15 EMG 3.0

First official release of EMG 3.

Index

B

Blacklist
    using 57

C

CMG SMSC 16
Connector states
    overview 26
Connectors
    incoming 25
    outgoing 25
    sample configurations 32
    short-circuit 25
    static 25

E

EBE
    exit code 44
emgd
    refreshing 21
    starting 21
    stopping 21
EMGDIR
    defined 19
Exit code
    when using EBE connector 44

F

Failover
    connectors 38

G

general
    log file 53
GSM 3.40
    User-Data Header 14

H

HTTP
    receiving messages 47
    sending messages 42

L

Load balancing
    connectors 38
Log files
    connector files 53
    format 54
    general 53
    security 53
Logging
    log level 53
    performance 53
Logica SMSC 16

M

Mappings
    incoming 28
    outgoing 28
    using 27
Messages
    binary 14

N

Nokia SMSC 16

O

Orphans
  overview 15
ORPHANSSIZE 15
OTA
  example sending 45

P

PARSEMESSAGE 96

R

Ringtone
  example sending 44
ROUTE
  connector keyword 36
  message option 36
ROUTELOGSIZE 15
Routing
  concatenated messages 41
  keyword-based 39, 96
  on prefix 37
  user-based 38, 107
Routing log
  defined 15
  overview 15
Routing table
  using 37
  using prefixes 37

S

Security
  access control 57
  blacklist 57
  overview 57
SMTP
  receiving messages 47
  sending messages 42

U

UDH
  long messages 14